9.1

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 1411.

To wydanie dotyczy wersji 9 wydania 1 produktu IBM® MQ oraz wszystkich kolejnych wydań i modyfikacji, o ile nie podano inaczej w nowych edycjach.

Wysyłając informacje do IBM, użytkownik przyznaje IBM niewyłączne prawo do używania i rozpowszechniania informacji w dowolny sposób, jaki uzna za właściwy, bez żadnych zobowiązań wobec ich autora.

Spis treści

Tworzenie aplikacji dla składnika IBM MQ

Istnieje możliwość tworzenia aplikacji w celu wysyłania i odbierania komunikatów oraz do zarządzania menedżerami kolejek i powiązanymi zasobami. Produkt IBM MQ obsługuje aplikacje napisane w wielu różnych językach i w różnych ramach.

Nowość w celu tworzenia aplikacji dla produktu IBM MQ?

Aby uzyskać informacje na temat tworzenia aplikacji dla produktu IBM MQ, należy odwiedzić produkt IBM Developer:

- LearnMQ *(podstawowe informacje, uruchamianie dema, tworzenie kodu aplikacji, zaawansowane kursy)*
- Pliki dotyczące produktu MQ do pobrania, przeznaczone dla programistów *(w tym bezpłatne edycje dla programistów i wersje próbne)*

W przypadku zapoznania się z pojęciami opisanymi w poniższych sekcjach można również łatwiej rozwijać aplikacje.

- “Pojęcia związane z projektowaniem aplikacji” na stronie 7
- “Uwagi dotyczące projektowania aplikacji produktu IBM MQ” na stronie 50

Obsługa języków i środowisk zorientowanych obiektom

Produkt IBM MQ udostępnia podstawowe funkcje obsługi aplikacji opracowanych w następujących językach i w następujących środowiskach:

- JMS
- Java
- C++
- .NET
- ActiveX (deprecated; użyj .NET)

Patrz także “Aplikacje obiektowe” na stronie 14.

Produkt .NET obsługuje aplikacje opracowane w wielu językach. Aby zilustrować korzystanie z klas produktu IBM MQ dla produktu .NET w celu uzyskania dostępu do kolejek produktu IBM MQ , w dokumentacji produktu MQ znajdują się informacje dotyczące następujących języków:

- C# przykładowy kod i aplikacje przykładowe
- Aplikacje przykładowe C++
- Visual Basic przykładowe aplikacje

Patrz sekcja “Pisanie i wdrażanie programów IBM MQ .NET” na stronie 587.

V 9.1.2 V 9.1.1 Produkt IBM MQ obsługuje podstawowy moduł produktu .NET dla aplikacji w środowiskach Windows z poziomu 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 Standard” na stronie 568.

ULW Produkt IBM MQ obsługuje również interfejs API produktu MQ Light , który implementuje protokół OASIS AMQP 1.0 . Istnieją interfejsy API przesyłania komunikatów dla następujących języków:

- Node.js
- Ruby
- Java
- Python

- Maven (projekt szkieletowy; korzysta z interfejsu API Java)
- Gradle (projekt szkieletowy; korzysta z interfejsu API Java)

Patrz także “Tworzenie aplikacji klienckich AMQP” na stronie 780.

Następujące powiązania języka są dostarczane w postaci:

- Powiązanie go
- Implementacja interfejsu API JavaScript , która współpracuje z aplikacjami Node.js

Obsługa programowych interfejsów REST API

Produkt IBM MQ zapewnia obsługę następujących programowych interfejsów REST API w celu wysyłania i odbierania komunikatów:

- V 9.1.0 IBM MQ messaging REST API
- z/OS IBM z/OS Connect EE
- IBM Integration Bus
- IBM Brama DataPower

Więcej informacji na ten temat zawiera sekcja “Tworzenie aplikacji REST przy użyciu produktu IBM MQ” na stronie 799, a także kurs Pierwsze kroki z interfejsem REST API przesyłania komunikatów produktu IBM MQ w obszarze IBM MQ produktu IBM Developer. This tutorial includes examples in the following languages, provided as-is, for use with the IBM MQ messaging REST API:

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

Wsparcie dla języków programowania proceduralnego

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

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

Języki te korzystają z interfejsu kolejki komunikatów (MQI) w celu uzyskania dostępu do usług kolejkowania komunikatów. Patrz sekcja “Tworzenie aplikacji MQI za pomocą programu IBM MQ” na stronie 806. Należy zauważyć, że model obiektu IBM MQ , używany przez języki i środowiska zorientowane obiektowe, udostępnia dodatkowe funkcje, które nie są dostępne dla języków proceduralnych korzystających z interfejsu MQI.

Określanie nazwy aplikacji

ULW V 9.1.2

Przed IBM MQ 9.1.2można określić nazwę aplikacji w aplikacjach klienckich 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 53.

Zadania pokrewne

“Tworzenie aplikacji dla składnika MQ Telemetry” na stronie 1354

“Tworzenie aplikacji Microsoft Windows Communication Foundation przy użyciu produktu IBM MQ” na stronie 1379
Niestandardowy kanał programu Microsoft Windows Communication Foundation (WCF) dla IBM MQ wysyła i odbiera komunikaty między klientami i usługami WCF.

Odsyłacze pokrewne

“Tworzenie aplikacji dla składnika Managed File Transfer” na stronie 1333
Określ programy, które mają być uruchamiane z programem Managed File Transfer, z Apache Ant , z Managed File Transfer, z programami wyjściowymi użytkownika Managed File Transfer sterowaniem Managed File Transfer , przez umieszczanie komunikatów w kolejce komend agenta.

Pojęcia związane z projektowaniem aplikacji

Do pisania aplikacji IBM MQ można użyć wyboru języków proceduralnych lub obiektowych. Przed rozpoczęciem projektowania i pisania aplikacji produktu IBM MQ należy zapoznać się z podstawowymi pojęciami dotyczącymi produktu IBM MQ .

Informacje na temat typów aplikacji, które można zapisać w programie IBM MQ, można znaleźć w sekcji “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 aplikacji produktu IBM MQ” na stronie 50
Po zdecydowaniu, w jaki sposób aplikacje mogą korzystać z platform i środowisk, które są dostępne dla użytkownika, należy zdecydować, w jaki sposób korzystać z funkcji oferowanych przez produkt IBM MQ.

Działania, które mogą być wykonywane przez aplikacje

Istnieje możliwość tworzenia aplikacji w celu wysyłania i odbierania komunikatów, które są potrzebne do obsługi procesów biznesowych. Istnieje również możliwość tworzenia aplikacji do zarządzania menedżerami kolejek i powiązanymi zasobami.

Działania, które mogą być wykonywane przez aplikacje w systemie IBM MQ for Multiplatforms

Multi

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

- Wysyłanie komunikatów do innych aplikacji działających w ramach tych samych systemów operacyjnych. Aplikacje mogą znajdować się na tym samym lub innym systemie.
- Wysyłanie komunikatów do aplikacji, które działają na innych platformach IBM MQ .
- Użyj kolejkowania komunikatów z programu CICS dla następujących systemów:
  - AIX TXSeries dla AIX
  - IBM i IBM i
  - Solaris Solaris
  - Windows Windows
- Użyj kolejkowania komunikatów z Encina dla systemów followania:
  - AIX AIX
  - Solaris Solaris
  - Windows Windows

- Użyj kolejkowania komunikatów z programu Tuxedo dla następujących systemów:
  - AIX
  - W & yś
  - Solaris
  - Windows
- Produkt IBM MQ należy używać jako menedżera transakcji, koordynując aktualizacje wykonywane przez zewnętrznych menedżerów zasobów w ramach jednostek pracy IBM MQ . Następujące zewnętrzne menedżery zasobów są obsługiwane i są zgodne z interfejsem XA X/OPEN.
  - Db2
  - Informix
  - Oracle
  - Sybase
- Przetwarzanie kilku komunikatów razem jako pojedyncza jednostka pracy, która może zostać zatwierdzona lub wycofana.
- Uruchom w pełnym środowisku produktu IBM MQ lub uruchom go z poziomu środowiska klienta IBM MQ .

Działania, które mogą być wykonywane przez aplikacje w systemie IBM MQ for z/OS

z/OS

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

- Użyj kolejkowania komunikatów w produkcie CICS lub IMS.
- Wysyłanie komunikatów między aplikacjami wsadowymi, CICSi IMS , wybierając najbardziej odpowiednie środowisko dla każdej funkcji.
- Wysyłanie komunikatów do aplikacji, które działają na innych platformach IBM MQ .
- Przetwarzanie kilku komunikatów razem jako pojedyncza jednostka pracy, która może zostać zatwierdzona lub wycofana.
- Wysyłanie komunikatów do aplikacji IMS i interakcja z nim za pomocą mostu IMS .
- Udział w jednostkach pracy koordynowanych przez RRS.

Każde środowisko w produkcie z/OS ma własne cechy, 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 korzyści płynnych z każdego środowiska. Na przykład można utworzyć interfejsy użytkownika końcowego za pomocą TSO lub CICS, można uruchomić moduły intensywnie przetwarzając w zadaniu wsadowym z/OS , a aplikacje bazy danych można uruchamiać w produkcie IMS lub CICS. We wszystkich przypadkach różne części aplikacji mogą komunikować się za pomocą komunikatów i kolejek.

Projektanci aplikacji produktu IBM MQ muszą mieć świadomość różnic i ograniczeń narzuconych przez te środowiska. Na przykład:

- Produkt IBM MQ udostępnia narzędzia umożliwiające komunikację między menedżerami kolejek (jest to nazywane *kolejkowaniem rozproszonym*).
- Metody zatwierdzania i wycofywania zmian różnią się między środowiskami zadań wsadowych i CICS .
- Produkt IBM MQ for z/OS zapewnia obsługę w środowisku produktu IMS dla programów przetwarzania komunikatów w trybie z połączeniem (MPPs), interaktywnych programów szybkiej ścieżki (IFPs) oraz programów przetwarzania komunikatów wsadowych (Batch Message processing programs-BMP). If you are writing batch DL/I programs, follow the guidance given in topics such as “Budowanie aplikacji wsadowych produktu z/OS” na stronie 1130 and “Uwagi dotyczące zadań wsadowych z/OS” na stronie 821 for z/OS batch programs.

- Mimo że w jednym systemie z/OS może istnieć wiele instancji produktu IBM MQ for z/OS , region CICS może jednocześnie łączyć się z tylko jednym menedżerem kolejek. Jednak więcej niż jeden region produktu CICS może być połączony z tym samym menedżerem kolejek. W środowiskach wsadowych 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, co zapewnia lepszą przepustowość i dostępność. Takie kolejki są nazywane *kolejkami współużytkowanymi*, a menedżery kolejek tworzą *grupę współużytkowania kolejek*, która może przetwarzać komunikaty w tych samych współużytkowanych kolejkach. Aplikacje wsadowe mogą łączyć się z jednym z kilku menedżerów kolejek w ramach grupy współużytkowania kolejek, określając nazwę grupy współużytkowania kolejki zamiast określonej nazwy menedżera kolejek. Jest ona znana jako *dołączanie wsadowe grupy*lub po prostu *przyłączenie grupy*. Więcej informacji zawiera sekcja Kolejki współużytkowane i grupy współużytkowania kolejek.

z/OS Różnice między obsługiwanymi środowiskami i ich ograniczeniami zostały opisane w dalszej części produktu “Używanie i zapisywanie aplikacji w systemie IBM MQ for z/OS” na stronie 983.

Pojęcia pokrewne

“Pojęcia związane z projektowaniem aplikacji” na stronie 7
Do pisania aplikacji IBM MQ można użyć wyboru języków proceduralnych lub obiektowych. Przed rozpoczęciem projektowania i pisania aplikacji produktu IBM MQ należy zapoznać się z podstawowymi pojęciami dotyczącymi produktu IBM MQ .

“Uwagi dotyczące projektowania aplikacji produktu IBM MQ” na stronie 50
Po zdecydowaniu, w jaki sposób aplikacje mogą korzystać z platform i środowisk, które są dostępne dla użytkownika, należy zdecydować, w jaki sposób korzystać z funkcji oferowanych przez produkt IBM MQ.

“Pisanie aplikacji proceduralnej w celu kolejkowania” na stronie 810
Ta sekcja zawiera informacje na temat pisania aplikacji kolejkowania, łączenia się i rozłączania z menedżerem kolejek, publikowania/subskrypcji oraz obiektów otwierających i zamykających.

“Pisanie aplikacji proceduralnych klienta” na stronie 1008
Co należy wiedzieć, aby pisać aplikacje klienckie w systemie IBM MQ , korzystając z języka proceduralnego.

“użycieIBM MQ classes for JMS” na stronie 83
IBM MQ classes for Java Message Service (IBM MQ classes for JMS) jest dostawcą JMS dostarczonym z produktem IBM MQ. Oprócz implementowania interfejsów zdefiniowanych w pakiecie javax.jms produkt IBM MQ classes for JMS udostępnia dwa zestawy rozszerzeń do interfejsu API produktu JMS .

“Korzystanie z interfejsu modelu obiektu komponentu (klasy automatyzacji produktuIBM MQ dla elementu ActiveX)” na stronie 686
Klasy automatyzacji produktu IBM MQ dla elementów ActiveX (MQAX) to komponenty ActiveX , które udostępniają klasy, których można używać w aplikacji w celu uzyskania dostępu do produktu IBM MQ.

“użycieIBM MQ classes for Java” na stronie 343
Użyj produktu IBM MQ w środowisku Java . Program IBM MQ classes for Java umożliwia aplikacji Java łączenie się z serwerem IBM MQ jako klient IBM MQ lub bezpośrednie połączenie z menedżerem kolejek produktu IBM MQ .

“Tworzenie aplikacji produktu .NET” na stronie 566
Program IBM MQ classes for .NET umożliwia programowi napisanego w środowisku programowania .NET nawiązanie połączenia z serwerem IBM MQ jako IBM MQ MQI client lub nawiązanie bezpośredniego połączenia z serwerem IBM MQ .

“Tworzenie aplikacji C++” na stronie 537
Produkt IBM MQ udostępnia klasy języka C + + równoważne obiektom produktu IBM MQ , a niektóre dodatkowe klasy są równoważne z typami danych tablicowych. Udostępnia ona wiele funkcji, które nie są dostępne w interfejsie MQI.

“Budowanie aplikacji proceduralnej” na stronie 1099
Aplikację IBM MQ można napisać w jednym z kilku języków proceduralnych, a następnie uruchomić aplikację na kilku różnych platformach.

Zadania pokrewne

“Korzystanie z przykładowych programów proceduralnych produktu IBM MQ” na stronie 1165
Te przykładowe programy są zapisywane w językach proceduralnych i demonstrują typowe zastosowania interfejsu kolejki komunikatów (Message Queue Interface-MQI). Programy IBM MQ na różnych platformach.

“Tworzenie aplikacji Microsoft Windows Communication Foundation przy użyciu produktu IBM MQ” na stronie 1379
Niestandardowy kanał programu Microsoft Windows Communication Foundation (WCF) dla 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. V 9.1.2 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 .

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.

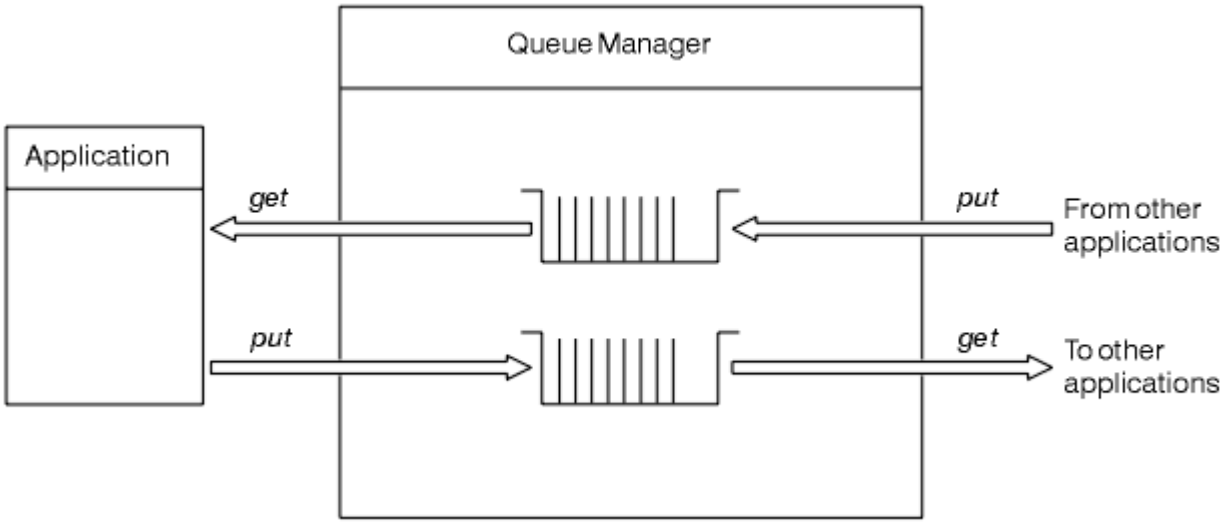
Pojęcia pokrewne

“Określanie nazwy aplikacji w obsługiwanych językach programowania” na stronie 53
Przed IBM MQ 9.1.2można już określić nazwę aplikacji w aplikacjach klienckich Java lub JMS .
W produkcie IBM MQ 9.1.2 ta funkcja jest rozszerzana na inne języki programowania w systemie IBM MQ for Multiplatforms.

Aplikacje korzystające z interfejsu MQI

Programy aplikacji 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

Podczas gdy aplikacje mogą umieszczać komunikaty w kolejkach lokalnych lub zdalnych (za pomocą komendy MQPUT), mogą one pobrać tylko komunikaty bezpośrednio z kolejek lokalnych (za pomocą komendy MQGET).

Zanim będzie można uruchomić tę aplikację, muszą zostać spełnione następujące warunki:

- Menedżer kolejek musi istnieć i być uruchomiony.
- Pierwsza kolejka aplikacji, z której komunikaty mają zostać usunięte, musi być zdefiniowana.
- Druga kolejka, na której aplikacja umieszcza komunikaty, musi być również zdefiniowana.
- Aplikacja musi być w stanie połączyć się z menedżerem kolejek. Aby to zrobić, musi być on połączony z produktem IBM MQ. Patrz “Budowanie aplikacji proceduralnej” na stronie 1099.
- Aplikacje, które umieszczają komunikaty w pierwszej kolejce, muszą również łączyć się z menedżerem kolejek. Jeśli są one zdalne, muszą być również skonfigurowane z kolejkami transmisji i kanałami. Ta część systemu nie jest wyświetlana w programie Rysunek 1 na stronie 11.

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

W jaki sposób można używać tabel definicji kanałów klienta (CCDTs), list nazw połączeń (lista CONNAME), równoważenia obciążenia i kodów pośredniczących kodu, aby połączyć się z wieloma menedżerami kolejek wraz z zaletami i niedogodami dla poszczególnych opcji dla konkretnego wymagania.

W poniższym tekście:

CCDT-multi-QMGR
Oznacza plik CCDT, który zawiera wiele kanałów połączenia klienckiego (CLNTCONN) z tą samą grupą, to jest atrybut połączenia klienta nazwy menedżera kolejek (QMNAME CLNTCONN), gdzie różne pozycje CLNTCONN są rozstrzygane do różnych menedżerów kolejek.

Jest to odrębne od pliku CCDT, który zawiera wiele pozycji CLNTCONN, które są po prostu różnymi adresami IP lub nazwami hostów dla tego samego menedżera kolejek z wieloma instancjami, co jest podejściem, które można wybrać w celu połączenia z kodem pośredniczonym kodu.

Jeśli zostanie wybrane podejście menedżera kolejek z wieloma kolejkami CCDT, należy wybrać, czy mają być priorytetyzowane pozycje, czy też mają być używane do losowego zarządzania obciążeniem (WLM):

Z priorytetem
Aby pamiętać o ostatnim dobrym połączeniu, należy użyć wielu pozycji uporządkowanych alfabetycznie z atrybutami CLNTWGHT (1) i AFFINITY (PREFERRED).

Randomizowane
Użyj atrybutów CLNTWGHT (1) i AFFINITY (NONE). Można dopasować wagę WLM dla różnych skalowanych serwerów IBM MQ , dopasując wartość CLNTWGHT.

Uwaga: Należy unikać dużych różnic w CLNTWGHT między kanałami.

System równoważenia obciążenia
Oznacza urządzenie sieciowe z wirtualnym adresem IP (VIP) skonfigurowane z monitorowaniem portów nasłuchiwania TCP/IP wielu menedżerów kolejek produktu IBM MQ . Sposób konfiguracji VIP w urządzeniu sieciowym zależy od używanego urządzenia sieciowego.

Poniższe opcje odnoszą się tylko do aplikacji wysyłających komunikaty lub inicjowania synchronicznego przesyłania komunikatów i odpowiedzi. Rozważania dotyczące aplikacji obsługujących te komunikaty i żądania, na przykład programy nasłuchujące są całkowicie oddzielne i szczegółowo omówione w sekcji "Łączenie obiektu nasłuchiwania komunikatów z kolejką" (TBD)

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

Lista CONNAME, CCDT multi-QMGR i Load Balancer
MQCONN ("NAZWA_MENEDŻERA_KOLEJEK") do MQCONN ("*QMNAME")

Nazwa menedżera kolejek może znajdować się w konfiguracji interfejsu JNDI (Java Naming and Directory Interface) dla aplikacji Java Enterprise Edition (EE). W przeciwnym razie wymagany jest jeden kod znaku-zmiana.

Jest to prawdopodobne, że będzie to miało pozytywny wpływ na kod.

Kod pośredniczący kodu
Zastąp istniejącą logikę połączenia JMS lub MQI kodem pośredniczonym kodu.

Jest to prawdopodobne, że będzie miało negatywny wpływ na kod.

Obsługa różnych strategii zarządzania obciążeniem

Lista CONNAME
Tylko priorytetyzowane.

Jest to prawdopodobne, że będzie miało negatywny wpływ na kod.

CCDT multiQMGR
Priorytetyzowane lub losowe.

Nie jest to prawdopodobne, aby miało to jakikolwiek wpływ na kod.

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

Jest to prawdopodobne, że będzie to miało pozytywny wpływ na kod.

Kod pośredniczący kodu
Dowolne, w tym każdy komunikat dla wszystkich komunikatów.

Jest to prawdopodobne, że będzie to miało pozytywny wpływ na kod.

Narzut wydajności podczas gdy podstawowy menedżer kolejek jest niedostępny

Lista CONNAME
Zawsze stara się najpierw na liście.

Jest to prawdopodobne, że będzie miało negatywny wpływ na kod.

CCDT multiQMGR
Pamięta ostatnie dobre połączenie.

Jest to prawdopodobne, że będzie to miało pozytywny wpływ na kod.

System równoważenia obciążenia
Monitorowanie portów pozwala uniknąć błędnych menedżerów kolejek.

Jest to prawdopodobne, że będzie to miało pozytywny wpływ na kod.

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

Jest to prawdopodobne, że będzie to miało pozytywny wpływ na kod.

Obsługa transakcji XA

Lista CONNAME, CCDT multi-QMGR i Load Balancer
Menedżer transakcji musi przechowywać informacje o odtwarzaniu, które łączą się ponownie z tym samym zasobem menedżera kolejek.

Wywołanie MQCONN, które jest tłumaczone na różne menedżery kolejek, generalnie unieważnia ten błąd. Na przykład w produkcie Java EE pojedyncza fabryka połączeń powinna zostać rozstrzygana do pojedynczego menedżera kolejek w przypadku korzystania z interfejsu XA.

Jest to prawdopodobne, że będzie miało negatywny wpływ na kod.

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

Jest to prawdopodobne, że będzie to miało pozytywny wpływ na kod.

Obsługa różnych strategii zarządzania obciążeniem

Lista CONNAME
Tylko serwer DNS.

Jest to prawdopodobne, że będzie miało negatywny wpływ na kod.

CCDT multiQMGR
System DNS i współużytkowany system plików, współużytkowany system plików lub plik CCDT.

Nie jest to prawdopodobne, aby miało to jakikolwiek wpływ na kod.

System równoważenia obciążenia
Dynamiczny wirtualny adres IP (VIP).

Jest to prawdopodobne, że będzie to miało pozytywny wpływ na kod.

Kod pośredniczący kodu
Pozycje CCDT menedżera kolejek lub pojedynczego menedżera kolejek.

Nie jest to prawdopodobne, aby miało to jakikolwiek wpływ na kod.

Unikanie zakłóceń związanych z planowaną konserwacją

Istnieje inna sytuacja, którą należy rozważyć i zaplanować, czyli w jaki sposób uniknąć zakłóceń w aplikacjach, na przykład błędów i przekroczeń limitu czasu widocznych dla użytkowników końcowych, podczas planowanej konserwacji menedżera kolejek. Najlepszym rozwiązaniem w celu uniknięcia zakłóceń jest usunięcie wszystkich prac z menedżera kolejek przed jego zatrzymaniem.

Rozważmy scenariusz żądania i odpowiedzi. Chcesz, aby wszystkie żądania w trakcie realizacji były zakończone, a odpowiedzi były przetwarzane przez aplikację, ale nie chcesz, aby dodatkowe prace zostały wprowadzone do systemu. Po prostu wyciszanie menedżera kolejek nie spełnia tej potrzeby, ponieważ poprawnie zakodowane aplikacje otrzymują kod powrotu RC2161 MQRC_Q_MGR_QUIESCING, zanim otrzymają komunikaty odpowiedzi dla żądań w trakcie realizacji.

Parametr PUT (DISABLED) można ustawić w kolejkach żądań używanych do wprowadzania pracy, pozostawiając jednocześnie kolejki odpowiedzi PUT (ENABLED) i GET (ENABLED). W ten sposób można monitorować głębokość kolejek żądań, transmisji i odpowiedzi. Po ustabilizowaniu się, czyli po zakończeniu lub wyczerpaniu żądań w trakcie realizacji, menedżer kolejek może zostać zatrzymany.

Jednak w celu obsługi kolejki żądań PUT (DISABLED) wymagane jest, aby podczas próby wysłania komunikatu wystąpił błąd w kolejce żądań PUT (DISABLED), której wynikiem jest kod powrotu RC2051 MQRC_PUT_INHIBITED (błąd powrotu_zbloczni_mqrc_put_inhibited).

Należy pamiętać, że wyjątek ten nie występuje podczas tworzenia połączenia z serwerem IBM MQlub otwierania kolejki żądań. Ten wyjątek występuje tylko wtedy, gdy podejmowana jest próba wysłania komunikatu za pomocą wywołania MQPUT.

Budowanie kodu pośredniczącego zawierającego tę logikę obsługi błędów dla scenariuszy żądań i odpowiedzi oraz żądanie, aby zespoły aplikacji używały takiego kodu pośredniczącego w przyszłości, mogą pomóc w opracowaniu aplikacji z zachowaniem spójnego zachowania.

Aplikacje obiektowe

Produkt IBM MQ zapewnia obsługę elementów JMS, Java, C + +, .NETi ActiveX. Te języki i środowiska używają modelu obiektów IBM MQ , który udostępnia klasy udostępniające te same funkcje, co wywołania i struktury produktu IBM MQ .

Niektóre języki i środowiska, które używają modelu obiektów IBM MQ , udostępniają dodatkowe funkcje, które nie są dostępne w przypadku korzystania z 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 15.

JMS
Produkt IBM MQ udostępnia klasy, które implementują specyfikację Java Message Service (JMS). Szczegółowe informacje na temat IBM MQ classes for JMSzawiera sekcja Korzystanie z programu IBM MQ classes for JMS. Informacje na temat różnic między IBM MQ classes for Java a IBM MQ classes for JMS , które ułatwiają podjęcie decyzji o ich użyciu, zawiera sekcja “Tworzenie aplikacji JMS i Java” na stronie 82.

Produkty IBM Message Service Client for C/C++ i IBM Message Service Client for .NET udostępniają aplikacyjny interfejs programistyczny (API) o nazwie XMS , który ma taki sam zestaw interfejsów co Java Message Service (JMS) API. Więcej informacji na ten temat zawiera sekcja “Tworzenie aplikacji produktu XMS .NET” na stronie 622.

Java
Informacje na temat kodowania programów za pomocą modelu obiektów IBM MQ w programie Javamożna znaleźć w sekcji Korzystanie z programu IBM MQ classes for Java . Program IBM

nie rozszerzy IBM MQ classes for Java i nie będzie on funkcjonalnie stabilizowany na poziomie dostarczonym w produkcie IBM MQ 8.0. Informacje na temat różnic między IBM MQ classes for Java i IBM MQ classes for JMS , które ułatwiają podjęcie decyzji o ich użyciu, można znaleźć w sekcji “Tworzenie aplikacji JMS i Java” na stronie 82.

C++

Produkt IBM MQ udostępnia klasy języka C + + równoważne obiektom produktu IBM MQ , a niektóre dodatkowe klasy są równoważne z typami danych tablicowych. Udostępnia ona wiele funkcji, które nie są dostępne w interfejsie MQI. Informacje Korzystanie z języka C++ temat kodowania programów za pomocą modelu obiektowego IBM MQ w języku C + +. Klienty usługi komunikatów dla środowisk C/C++ i .NET zawierają interfejs API o nazwie XMS , który zawiera ten sam zestaw interfejsów co Java Message Service (JMS). API.

.NET

Informacje na temat kodowania programów .NET przy użyciu klas IBM MQ .NET znajdują się w sekcji Tworzenie aplikacji .NET . Klienty usługi komunikatów dla środowisk C/C++ i .NET udostępniają aplikacyjny interfejs programistyczny (API) o nazwie XMS , który ma taki sam zestaw interfejsów co Java Message Service (JMS) API.

ActiveX

Element IBM MQ ActiveX jest powszechnie znany jako MQAX. Produkt MQAX jest dołączony jako część produktu IBM MQ for Windows. Obsługa elementu ActiveX została ustabilizowana na poziomie IBM WebSphere MQ 6.0 . Informacje na temat kodowania programów za pomocą modelu obiektu IBM MQ w podręczniku ActiveX Korzystanie z interfejsu Component Object Model Interface (WebSphere MQ Automation Classes for ActiveX).

W produkcie IBM MQ 9.0obsługa produktu IBM MQ dla produktu Microsoft Active X jest nieaktualna. Zalecaną technologią zastępczą są klasy IBM MQ dla .NET. Więcej informacji na ten temat zawiera sekcja Tworzenie aplikacji .NET.

Pojęcia pokrewne

“Tworzenie aplikacji MQI za pomocą programu IBM MQ” na stronie 806
Produkt IBM MQ udostępnia 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 wyboru języków proceduralnych lub obiektowych. Przed rozpoczęciem projektowania i pisania aplikacji produktu IBM MQ należy zapoznać się z podstawowymi pojęciami dotyczącymi produktu IBM MQ .

Odsyłacze pokrewne

Informacje dodatkowe dotyczące programowania aplikacji

Model obiektu IBM MQ

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

Model obiektu IBM MQ składa się z następujących elementów:

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

Podczas tworzenia aplikacji IBM MQ przy użyciu modelu obiektów IBM MQ tworzone są instancje tych klas w aplikacji. Instancja klasy w programowaniu obiektowym jest nazywana *obiektem*. Gdy obiekt został utworzony, użytkownik wchodzi w interakcję z obiektem przez sprawdzenie lub ustawienie wartości właściwości obiektu (odpowiednik wywołania wywołania MQINQ lub MQSET) oraz przez wywołanie metody w odniesieniu do obiektu (odpowiednik wydania innych wywołań MQI).

Klasy

Model obiektu 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. Ma metody do łączenia (), Disconnect (), Commit () i Backout () (odpowiednik MQCONN lub MQCONNX, MQDISC, MQCMIT i MQBACK). Ma ona właściwości odpowiadające atrybutom menedżera kolejek. Uzyskiwanie dostępu do właściwości atrybutu menedżera kolejek niejawnie łączy się z menedżerem kolejek, jeśli nie jest jeszcze połączony. Zniszczenie obiektu MQQueueManager w sposób niejawny rozłącza się z menedżerem kolejek.

MQQUEUE
Obiekt klasy MQQueue reprezentuje kolejkę. Posiada metody umieszczania () i Get () komunikatów do i z kolejki (odpowiedniki 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 komendy MQOPEN). Zniszczenie obiektu MQQueue powoduje niejawne zamknięcie kolejki (odpowiednik komendy MQCLOSE).

Temat MQTopic
Obiekt klasy MQTopic reprezentuje temat. Posiada metody umieszczania (publikowania) i Get () (odbieranie lub subskrybowanie) komunikatów do i z tematu (równoważnego wywołania MQPUT i MQGET). Ma właściwości odpowiadające atrybutom tematu. Dostęp do obiektu MQTopic można uzyskać tylko w przypadku publikacji lub subskrypcji, a nie obu jednocześnie. W przypadku użycia do odbierania komunikatów obiekt MQTopic może zostać utworzony przy użyciu niezarządzanej lub zarządzanej subskrypcji oraz jako trwały lub nietrwały subskrybent-dla tych różniących się scenariuszy udostępniono wiele przeciążonych konstruktorów.

Komunikat MQMessage
Obiekt klasy MQMessage reprezentuje komunikat, który ma zostać umieszczony w kolejce lub został zwrócony z kolejki. Zawiera on bufor i hermetykuje zarówno dane aplikacji, jak i MQMD. Ma on właściwości odpowiadające pola MQMD oraz metody umożliwiające zapisywanie i odczytywanie danych użytkownika różnych typów (na przykład łańcuchów, długich liczb całkowitych, krótkich liczb całkowitych, pojedynczych bajtów) do buforu i z niego.

Opcje MQPutMessage
Obiekt klasy opcji MQPutMessagereprezentuje strukturę MQPMO. Ma właściwości odpowiadające polom MQPMO.

Opcje MQGetMessage
Obiekt klasy MQGetMessageOptions reprezentuje strukturę MQGMO. Ma właściwości odpowiadające polu MQGMO.

Proces MQProcess
Obiekt klasy MQProcess reprezentuje definicję procesu (używaną z wyzwalaniem). Zawiera właściwości reprezentujące atrybuty definicji procesu.

Multi **MQDistributionList**
Obiekt klasy MQDistributionList reprezentuje listę dystrybucyjną (używaną do wysyłania wielu komunikatów za pomocą jednej operacji MQPUT). Zawiera ona listę obiektów elementów MQDistributionList.

Multi **Element MQDistributionList**
Obiekt klasy elementu MQDistributionListreprezentuje miejsce docelowe pojedynczej listy dystrybucyjnej. Obudowuje struktury MQOR, MQRR i MQPMR i ma właściwości odpowiadające danym dziedzinom tych struktur.

odwołania do obiektów

W programie IBM MQ , który korzysta z interfejsu MQI, program IBM MQ zwraca uchwyty połączeń i uchwyty obiektów do programu.

Te uchwyty muszą być przekazywane jako parametry w kolejnych wywołaniach produktu IBM MQ . W przypadku modelu obiektu IBM MQ uchwyty te są ukryte w programie użytkowym. Zamiast tego tworzenie obiektu z klasy powoduje, że odwołanie do obiektu jest zwracane do programu aplikacji. Jest to odwołanie do obiektu, które jest używane podczas wykonywania wywołań metod i dostępu do właściwości dla obiektu.

Kody powrotu

Wywołanie metody wywołania metody lub ustawienie wartości właściwości powoduje ustawienie zwracanych kodów powrotu.

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

Komunikaty produktu IBM MQ

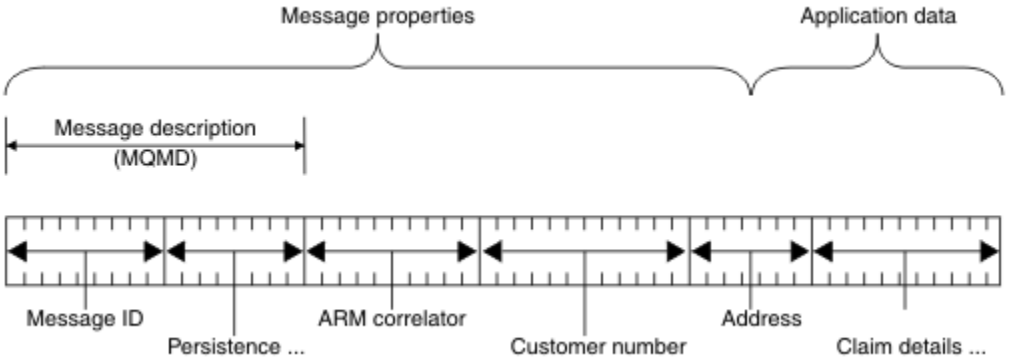
Komunikat IBM MQ składa się z właściwości komunikatu i danych aplikacji. Deskryptor komunikatu kolejkowania komunikatów (MQMD) zawiera informacje sterujące, które towarzyszy danych aplikacji podczas przesyłania komunikatów między aplikacjami wysyłającą i odbierającą.

Części komunikatu

Komunikaty produktu IBM MQ składają się z dwóch części:

- Właściwości komunikatu
- Dane aplikacji

Rysunek 2 na stronie 17 reprezentuje komunikat i pokazuje, w jaki sposób jest logicznie podzielony na właściwości komunikatu i dane aplikacji.



Rysunek 2. Reprezentacja komunikatu

Dane aplikacji, które są przenoszone w komunikacie IBM MQ , nie są zmieniane przez menedżer kolejek, jeśli nie zostanie przeprowadzona na nim konwersja danych. Ponadto produkt IBM MQ nie ma żadnych ograniczeń dotyczących treści tych danych. Długość danych w każdym komunikacie nie może przekraczać wartości atrybutu **MaxMsgLength** zarówno w menedżerze kolejek, jak i w menedżerze kolejek.

ULW W systemie UNIX, Linux, and Windowsatrybut *MaxMsgLength* menedżera kolejek i wartość domyślna kolejki to 4 MB (4 194 304 bajtów), które można zmienić maksymalnie o 100 MB (104 857 600 bajtów), jeśli jest to wymagane.

IBM i W systemie IBM iatrybut *MaxMsgLength* menedżera kolejek i wartość domyślna kolejki to 4 MB (4 194 304 bajtów), które można zmienić maksymalnie o 100 MB (104 857 600 bajtów), jeśli jest to wymagane. Jeśli planowane jest używanie komunikatów programu IBM MQ większych niż 15 MB w systemie IBM i, należy zapoznać się z “Budowanie aplikacji proceduralnej w systemie IBM i” na stronie 1106.

z/OS W systemie z/OSatrybut **MaxMsgLength** menedżera kolejek jest ustalony na 100 MB, a atrybut **MaxMsgLength** w kolejce jest domyślnie ustawiony na 4 MB (4 194 304 bajtów), co można zmienić maksymalnie o 100 MB, jeśli jest to wymagane.

W niektórych okolicznościach komunikaty są nieco krótsze niż wartość atrybutu **MaxMsgLength** . Więcej informacji na ten temat zawiera sekcja “Dane w komunikacie” na stronie 849.

Komunikat tworzy się podczas korzystania z wywołań MQI MQPUT lub MQPUT1 . Jako dane wejściowe dla tych wywołań należy podać informacje sterujące (takie jak priorytet komunikatu i nazwę kolejki odpowiedzi) oraz dane użytkownika, a następnie wywołanie powoduje umieszczenie komunikatu w kolejce. Więcej informacji na temat tych wywołań można znaleźć w sekcji MQPUT i MQPUT1 .

deskryptor komunikatu

Dostęp do informacji dotyczących kontroli komunikatów można uzyskać za pomocą struktury MQMD, która definiuje *deskryptor komunikatu*.

Pełny opis struktury MQMD znajduje się w sekcji MQMD-deskryptor komunikatu.

Opis sposobu korzystania z pól w strukturze MQMD, które zawierają informacje o pochodzeniu komunikatu, zawiera sekcja “Kontekst komunikatu” na stronie 47.

Istnieją różne wersje deskryptora komunikatu. Dodatkowe informacje na temat grupowania i segmentacji komunikatów (patrz “Grupy komunikatów” na stronie 44) są dostępne w wersji 2 deskryptora komunikatu (lub MQMDE). Jest on taki sam, jak deskryptor komunikatu w wersji 1, ale ma dodatkowe pola. Te pola są opisane w sekcji MQMDE-Rozszerzenie deskryptora komunikatu.

Typy komunikatów

Istnieją cztery typy komunikatów zdefiniowane przez produkt IBM MQ.

Te cztery komunikaty są następujące:

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

Aplikacje mogą korzystać z pierwszych trzech typów komunikatów w celu przekazywania informacji między sobą. Czwarty typ raportu jest przeznaczony dla aplikacji i menedżerów kolejek, które mają być używane do raportowania informacji o zdarzeniach, takich jak wystąpienie błędu.

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

Datagramy

datagramu należy używać wtedy, gdy nie jest wymagana odpowiedź z aplikacji, która odbiera komunikat (to znaczy pobiera komunikat z kolejki).

Przykład aplikacji, która może używać datagramów, to taka, która wyświetla informacje o locie w salonie lotniskowym. Komunikat może zawierać dane dla całego ekranu informacji o locie. Taka aplikacja prawdopodobnie nie zażądała potwierdzenia dla komunikatu, ponieważ prawdopodobnie nie

ma znaczenia, czy komunikat nie został dostarczony. Po krótkim czasie aplikacja wysyła komunikat aktualizacji.

Komunikaty żądań

Użyj *komunikatu żądania* , jeśli chcesz uzyskać odpowiedź z aplikacji, która odbierze komunikat.

Przykład aplikacji, która może używać komunikatów żądań, to taka, która wyświetla saldo konta sprawdzanego. Komunikat żądania może zawierać numer konta, a komunikat odpowiedzi będzie zawierał saldo konta.

Jeśli chcesz powiązać komunikat odpowiedzi z komunikatem żądania, istnieją dwie opcje:

- Utwórz aplikację obsługując komunikat żądania, który jest odpowiedzialny za zapewnienie, że umieszcza informacje w komunikacie odpowiedzi, który odnosi się do komunikatu żądania.
- Użyj pola raportu w deskryptorze komunikatu żądania, aby określić treść pól *MsgId* i *CorrelId* w komunikacie odpowiedzi:
  - Można zażądać, aby *MsgId* lub *CorrelId* oryginalnego komunikatu był kopiowany do pola *CorrelId* komunikatu odpowiedzi (domyślnym działaniem jest kopiowanie elementu *MsgId*).
  - Można zażądać, aby dla komunikatu odpowiedzi został wygenerowany nowy *MsgId* , albo że element *MsgId* oryginalnego komunikatu ma zostać skopiowany do pola *MsgId* komunikatu odpowiedzi (domyślnym działaniem jest wygenerowanie nowego identyfikatora komunikatu).

Komunikaty odpowiedzi

W przypadku odpowiedzi na inny komunikat należy użyć *komunikatu odpowiedzi* .

Podczas tworzenia komunikatu odpowiedzi należy przestrzegać wszystkich opcji, które zostały ustawione w deskryptorze komunikatu, do którego tworzona jest odpowiedź. Opcje raportu określają treść pól identyfikatora komunikatu (*MsgId*) oraz identyfikatora korelacji (*CorrelId*). Te pola umożliwiają aplikacji, która odbiera odpowiedź, w celu skorelowania odpowiedzi z jej oryginalnym żądaniem.

Komunikaty raportów

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

Mogą one być generowane przez:

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

Komunikaty raportów mogą być generowane w dowolnym momencie i mogą pojawić się w kolejce, gdy aplikacja nie oczekuje na nie.

Typy komunikatów raportu

Po umieszczniu komunikatu w kolejce można wybrać opcję odbierania:

- *Komunikat raportu o wyjątku*. Ta opcja jest wysyłana w odpowiedzi na komunikat z ustawioną flagą wyjątków. Jest on generowany przez agenta 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 o potwierdzeniu przybycia (COA)*. Oznacza to, że komunikat osiągnął swoją kolejkę docelową. Jest on generowany przez menedżer kolejek.
- Komunikat raportu *Potwierdzenie 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 (co oznacza, że działanie żądane w komunikacie zostało wykonane pomyślnie). Ten typ raportu jest generowany przez aplikację.
- *Komunikat raportu powiadomienia o działaniu negatywnym (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 jedną z następujących wartości:

- Cały oryginalny komunikat
- Pierwsze 100 bajtów danych w oryginalnym komunikacie
- Brak danych z oryginalnego komunikatu

W przypadku umieszczenia komunikatu w kolejce można zażądać więcej niż jednego typu komunikatu raportu. Jeśli zostanie wybrany komunikat o potwierdzeniu dostarczenia i opcje komunikatu raportu o wyjątku, jeśli komunikat nie zostanie dostarczony, zostanie wyświetlony komunikat o wyjątku. Jeśli jednak zostanie wybrana tylko opcja komunikatu z potwierdzeniem dostarczenia, a komunikat nie zostanie dostarczony, nie zostanie wyświetlony komunikat o raporcie o wyjątku.

Komunikaty raportu, które są wysyłane, gdy spełnione są kryteria generowania konkretnego komunikatu, są jedynymi otrzymanymi.

Opcje komunikatów raportu

Po wystąpieniu wyjątku można *odrzucić* komunikat. Jeśli wybrano opcję odrzucenia i zażądano komunikatu raportu o wyjątku, komunikat raportu będzie kierowany do serwerów *ReplyToQ* i *ReplyToQMgr*, a oryginalny komunikat zostanie usunięty.

Uwaga: Korzyścią z tego jest to, że można zmniejszyć liczbę komunikatów, które trafiają do kolejki niedostarczonych komunikatów. Oznacza to jednak, że aplikacja, o ile nie wysyła tylko komunikatów datagramu, ma do czynienia z zwrócanych wiadomości. Po wygenerowaniu komunikatu raportu o wyjątku dziedziczy on trwałość oryginalnego komunikatu.

Jeśli komunikat raportu nie może zostać dostarczony (jeśli kolejka jest pełna, na przykład), komunikat raportu jest umieszczany w kolejce niedostarczonych komunikatów.

Jeśli chcesz otrzymać komunikat raportu, podaj 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.

W deskryptorze komunikatu (MQMD) komunikatu można użyć innych opcji raportu, aby określić treść pól *MsgId* i *CorrelId* wszystkich komunikatów raportu utworzonych dla tego komunikatu:

- Użytkownik może zażądać skopiowania *MsgId* lub *CorrelId* oryginalnego komunikatu do pola *CorrelId* komunikatu raportu. Domyślne działanie polega na skopiowaniu identyfikatora komunikatu. Użyj identyfikatora MQRO_COPY_MSG_ID_TO_CORRELID, ponieważ umożliwia nadawcę komunikatu korelowanie komunikatu odpowiedzi lub raportu z oryginalnym komunikatem. Identyfikator korelacji komunikatu odpowiedzi lub komunikatu raportu jest identyczny z identyfikatorem komunikatu oryginalnego komunikatu.
- Użytkownik może zażądać, aby dla komunikatu raportu został wygenerowany nowy *MsgId* albo że *MsgId* oryginalnego komunikatu ma zostać skopiowany do pola *MsgId* komunikatu raportu. Domyślne działanie polega na wygenerowaniu nowego identyfikatora komunikatu. Użyj komendy MQRO_NEW_MSG_ID, ponieważ zapewnia on, że każdy komunikat w systemie ma inny identyfikator komunikatu i może zostać jednoznacznie odróżniony od wszystkich innych komunikatów w systemie.
- W przypadku aplikacji specjalizowanych może być konieczne użycie identyfikatora MQRO_PASS_MSG_ID lub MQRO_PASS_CORREL_ID. Należy jednak zaprojektować aplikację, która odczytuje komunikaty z kolejki, aby upewnić się, że działa poprawnie, gdy na przykład kolejka zawiera wiele komunikatów o tym samym identyfikatorze komunikatu.

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

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

Podczas generowania raportu na temat komunikatu, aplikacje serwera muszą sprawdzić, czy którekolwiek z tych opcji zostały ustawione.

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

Aby wskazać rodzaj raportu, menedżery kolejek używają zakresu kodów sprzężenia zwrotnego. Te kody umieszczają w polu *Feedback* deskryptora komunikatu komunikatu raportu. Menedżery kolejek mogą również zwracać kody przyczyny MQI w polu *Feedback* . Produkt IBM MQ definiuje zakres kodów sprzężenia zwrotnego dla aplikacji, które mają być używane.

Więcej informacji na temat informacji zwrotnych i kodów przyczyny można znaleźć w sekcji Opinia.

Przykładem programu, który może użyć kodu sprzężenia zwrotnego, jest monitorowanie obciążeń innych programów obsługujących kolejkę. Jeśli istnieje więcej niż jedna instancja programu obsługującego kolejkę, a liczba komunikatów przybywających do kolejki nie uzasadnia tego, taki program może wysłać komunikat raportu (z kodem sprzężenia zwrotnego MQFB_QUIT) do jednego z programów obsługujących, aby wskazać, że program powinien zakończyć działanie. (Program monitorujący może użyć wywołania MQINQ, aby dowiedzieć się, ile programów serwuje kolejkę).

Multi *Raporty i segmentowane komunikaty*

Nieobsługiwane w produkcie IBM MQ for z/OS.

Jeśli komunikat jest posegmentowany, a użytkownik prosi o wygenerowanie raportów, może otrzymać więcej raportów niż to, co zrobiłby, gdyby komunikat nie był posegmentowany.

Opis posegmentowanych komunikatów znajduje się w sekcji “Segmentacja komunikatów” na stronie 884.

Dla raportów wygenerowanych przez produkt IBM MQ

Jeśli komunikaty zostaną segmentowane lub zezwolono na to menedżer kolejek, istnieje tylko jedna obserwacja, w której można oczekiwać otrzymania pojedynczego raportu dla całego komunikatu. Jest to sytuacja, w której zażądano tylko raportów COD i określono MQGMO_COMPLETE_MSG dla aplikacji pobierających.

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

Uwaga: Jeśli komunikaty są segmentowane i wymagane jest tylko pierwsze 100 bajtów oryginalnych danych komunikatu, które mają zostać zwrócone, należy zmienić ustawienie opcji raportu, aby poprosić o raporty bez danych dla segmentów, które mają przesunięcie 100 lub więcej. Jeśli tego nie zrobisz, pozostaw to ustawienie tak, aby każdy segment żądał 100 bajtów danych, a użytkownik wczytał komunikaty raportu za pomocą pojedynczej komendy MQGET, podając MQGMO_COMPLETE_MSG, do dużego komunikatu zawierającego 100 bajtów danych odczytu na każdym odpowiednim przesuniętym przesunięciom. W takim przypadku potrzebny jest duży bufor lub konieczne jest podanie wartości MQGMO_ACCEPT_TRUNCATED_MSG.

Dla raportów generowanych przez aplikacje

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

Następnie do danych komunikatu raportu należy dodać wartość none (brak), 100 bajtów (lub wszystkie oryginalne dane komunikatu) (lub inną wartość, która zwykle jest dołączana).

Nagłówki IBM MQ , które należy skopiować, można rozpoznać po wyszukaniu kolejnych nazw formatów, zaczynając od deskryptora MQMD i kontynuując je przez wszystkie obecne nagłówki. Następujące nazwy `Format` wskazują następujące nagłówki IBM MQ :

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

MQH* oznacza dowolną nazwę rozpoczynający się od znaków MQH.

Nazwa `Format` występuje na 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 nagłówków MQMDE, MQIMSi wszystkich nagłówków MQH*.

Jeśli używany jest deskryptor MQMD w wersji 1, a użytkownik zgłasza segment lub komunikat w grupie lub komunikat, dla którego dozwolony jest segmentację, dane raportu muszą rozpoczynać się od wywołania MQMDE. Ustaw pole *`OriginalLength`* na długość oryginalnych danych komunikatu, z wyłączeniem długości znalezionych nagłówków IBM MQ .

Pobieranie raportów

Jeśli zapytasz o raporty COA lub COD, możesz poprosić o ich ponowne zmontowania dla Ciebie z MQGMO_COMPLETE_MSG.

Wywołanie MQGET z MQGMO_COMPLETE_MSG jest spełnione, gdy w kolejce znajdują się wystarczające komunikaty raportów (jednego typu, na przykład COA, i z tym samym *`GroupId`*), które reprezentują jeden kompletny oryginalny komunikat. Jest to prawda, nawet jeśli same komunikaty raportu nie zawierają pełnych danych pierwotnych; pole *`OriginalLength`* w każdym komunikacie raportu podaje długość pierwotnych danych reprezentowanych przez ten komunikat raportu, nawet jeśli same dane nie są obecne.

Tej techniki można użyć nawet wtedy, gdy w kolejce znajduje się kilka różnych typów raportów (na przykład COA i COD), ponieważ MQGET z MQGMO_COMPLETE_MSG zawiera komunikaty raportów tylko wtedy, gdy mają ten sam kod produktu *`Feedback`* . Nie można jednak używać tej techniki w przypadku raportów o wyjątkach, ponieważ w ogóle mają one różne kody *`Feedback`* .

Tej techniki można użyć do uzyskania pozytywnego wskazania, że cały komunikat dotarł. Jednak w większości przypadków konieczne jest zastosowanie niektórych segmentów w celu uzyskania pewnych segmentów, podczas gdy inne mogą wygenerować wyjątek (lub wygaśnięcie, o ile to możliwe). W tym przypadku nie można użyć komendy MQGMO_COMPLETE_MSG, ponieważ w ogólnym przypadku mogą być różne kody *`Feedback`* dla różnych segmentów, a w przypadku segmentu może być więcej niż jeden raport. Można jednak użyć komendy MQGMO_ALL_SEGMENTS_AVAILABLE.

Aby zezwolić na to, konieczne może być pobranie raportów w miarę ich przybycia, a następnie zbudowanie obrazu w aplikacji, co stało się z oryginalnym komunikatem. Można użyć pola *`GroupId`* w komunikacie raportu, aby skorelować raporty z *`GroupId`* oryginalnego komunikatu oraz pole *`Feedback`* , aby określić typ każdego komunikatu raportu. Sposób, w jaki to robisz, zależy od wymagań aplikacji.

Jedno podejście jest następujące:

- Zapytaj o raporty dotyczące COD i raporty o wyjątkach.
- Po określonym czasie sprawdź, czy kompletny zestaw raportów COD został odebrany za pomocą komendy MQGMO_COMPLETE_MSG. Jeśli tak, aplikacja wie, że cały komunikat został przetworzony.
- Jeśli nie, a raporty o wyjątkach odnoszące się do tego komunikatu są obecne, rozwiążesz problem w odniesieniu do nieposegmentowanych komunikatów, ale upewnij się, że w pewnym momencie wyczyszczasz segmenty osierote.

- Jeśli istnieją segmenty, dla których nie ma żadnych raportów, oryginalne segmenty (lub raporty) mogą oczekiwać na ponowne podłączenie kanału lub przeciążenia sieci w pewnym momencie. Jeśli w ogóle nie otrzymano żadnych raportów o wyjątkach (lub jeśli uważasz, że te, które mogą być tylko tymczasowe), możesz zdecydować się na to, aby aplikacja czekała nieco dłużej.

  Tak jak poprzednio, jest to podobne do rozważań, jakie masz przy kontaktach z nieposegmentowanymi wiadomościami, poza tym, że trzeba również rozważyć możliwość czyszczenia segmentów osierotnych.

Jeśli oryginalny komunikat nie jest krytyczny (na przykład jeśli jest to zapytanie lub komunikat, który może zostać powtórzony później), należy ustawić czas utraty ważności, aby upewnić się, że osierote segmenty są usuwane.

Menedżery kolejek na poziomie zaplecza

Gdy raport jest generowany przez menedżer kolejek, który obsługuje segmentację, ale jest odbierany w menedżerze kolejek, który nie obsługuje segmentacji, struktura MQMDE (identyfikująca *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, jeśli raport jest tam generowany, struktura MQMDE w pierwotnym komunikacie jest traktowana wyłącznie jako dane. W związku z tym nie jest ona uwzględniana w danych raportu, jeśli zażądano zerowej liczby bajtów danych pierwotnych. Bez wywołania MQMDE komunikat raportu może nie być przydatny.

Zażądaj co najmniej 100 bajtów danych w raportach, jeśli istnieje możliwość, że komunikat może przemieszczać się przez menedżer kolejek z powrotem na poziomie zaplecza.

Format informacji sterujących komunikatów i danych komunikatu

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

Format informacji o sterowaniu komunikatami

Informacje sterujące w polach łańcucha znaków w deskryptorze 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 kanałów komunikatów, które przesyłają komunikaty, używają wartości tego atrybutu w celu określenia, jakie dane mają być wykonywane.

Format danych komunikatu

Można określić dowolną z następujących czynności:

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

W tym celu należy użyć następujących pól:

Format
Oznacza to, że odbiorca komunikatu ma format danych aplikacji w komunikacie.

Gdy menedżer kolejek tworzy komunikat, w niektórych okolicznościach używa pola *Format* w celu zidentyfikowania formatu tego komunikatu. Na przykład, gdy menedżer kolejek nie może dostarczyć komunikatu, umieszcza komunikat w kolejce niedostarczonych komunikatów (niedostarczonych komunikatów). Dodaje nagłówek (zawierający więcej informacji sterujących) do komunikatu, a następnie zmienia pole *Format* , aby to pokazać.

Menedżer kolejek ma *wbudowane formaty* o nazwach rozpoczynających się od MQ, na przykład MQFMT_STRING. Jeśli nie są one zgodne z potrzebami użytkownika, można zdefiniować własne formaty (*formaty zdefiniowane przez użytkownika*), ale nie wolno używać nazw rozpoczynających się od MQ .

Podczas tworzenia własnych formatów i korzystania z nich należy napisać wyjście konwersji danych w celu obsługi programu pobieranego za pomocą komendy MQGMO_CONVERT.

CodedCharSetId
Definiuje zestaw znaków danych znakowych w komunikacie. Aby ustawić ten zestaw znaków na wartość tego menedżera kolejek, można ustawić to pole na stałe MQCCSI_Q_MGR lub MQCCSI_INHERIT.

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

Encoding
Opisuje format danych liczbowych komunikatu, który zawiera binarne liczby całkowite, liczby całkowite z upakowanymi liczbami całkowitymi i liczby zmiennopozycyjne. Zwykle jest ona zakodowana zgodnie z określoną maszyną, na której jest uruchomiony menedżer kolejek.

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

Po dostaniu komunikatu z kolejki należy porównać wartość pola *Encoding* w deskryptorze komunikatu z wartością stałej MQENC_NATIVE na komputerze. Jeśli te dwie wartości różnią się, 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 której występują różne platformy.

Może ona zostać przekształcona w wysyłającym menedżerze kolejek lub w odbierającym menedżerze kolejek. Jeśli biblioteka wbudowanych formatów nie spełnia Twoich potrzeb, możesz zdefiniować swoje własne. Typ konwersji zależy od formatu komunikatu określonego w polu formatu deskryptora komunikatu, MQMD.

Uwaga: Komunikaty z podanym parametrem MQFMT_NONE nie są przekształcane.

Konwersja w wysyłającym menedżerze kolejek

Ustaw atrybut kanału CONVERT na wartość YES, jeśli chcesz, aby wysyłający agent kanału komunikatów (MCA) przekształcił dane aplikacji.

Konwersja jest wykonywana w wysyłającym menedżerze kolejek dla niektórych wbudowanych formatów i dla formatów zdefiniowanych przez użytkownika, jeśli zostanie dostarczone odpowiednie wyjście użytkownika.

Wbudowane formaty

Są to:

- Komunikaty, które są znakami wszystkich znaków (przy użyciu nazwy formatu MQFMT_STRING)
- IBM MQ zdefiniowane komunikaty, na przykład Formaty komend programowalnych

  Produkt IBM MQ korzysta z komunikatów formatu komend programowalnych dla komunikatów administracyjnych i zdarzeń (w tym przypadku używana jest nazwa formatu MQFMT_ADMIN). Można użyć tego samego formatu (przy użyciu nazwy formatu MQFMT_PCF) dla własnych komunikatów, a także skorzystać z wbudowanej konwersji danych.

Wszystkie wbudowane formaty menedżera kolejek mają nazwy rozpoczynające się od wywołania 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 “Pisanie wyjść konwersji danych” na stronie 1082). W środowisku klient-serwer wyjście jest ładowane na serwerze, a konwersja odbywa się w tym miejscu.

Konwersja w odbierającym menedżerze kolejek

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

Konwersja jest wykonywana w trakcie przetwarzania wywołania MQGET, jeśli zostanie określona opcja MQGMO_CONVERT. Szczegółowe informacje na ten temat zawiera sekcja Opcje .

Kodowane zestawy znaków

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

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

W przypadku ogólnych konwersji danych, jeśli bazowy system operacyjny obsługuje strony kodowe DBCS, produkt IBM MQ może go używać.

Szczegółowe informacje na temat obsługiwanych kodowanych zestawów znaków można znaleźć w dokumentacji systemu operacyjnego.

Podczas pisania aplikacji, które obejmują wiele platform, należy wziąć pod uwagę konwersję danych aplikacji, nazwy formatów i wyjścia użytkownika. Informacje na temat wywoływania i zapisywania wyjść konwersji danych można znaleźć w sekcji “Pisanie wyjść konwersji danych” na stronie 1082 .

Priorytety komunikatów

Można ustawić priorytet komunikatu na wartość liczbową lub pozwolić na to, aby komunikat miał domyślny priorytet kolejki.

Priorytet komunikatu (w polu *Priority* struktury MQMD) ustawia się podczas umieszczania komunikatu w kolejce. Dla priorytetu można ustawić wartość liczbową lub pozwolić na to, aby komunikat był domyślnym priorytetem kolejki.

Atrybut **MsgDeliverySequence** kolejki określa, czy komunikaty w kolejce są przechowywane w sekwencji FIFO (najpierw w pierwszej kolejności), czy w FIFO w kolejności priorytetów. 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 zapisywane w kolejce w celu ich przybycia.

Atrybut **DefPriority** kolejki ustawia domyślną wartość priorytetu dla komunikatów umieszczanych w tej kolejce. Ta wartość jest ustawiana podczas tworzenia kolejki, ale może zostać zmieniona później. Kolejki aliasowe i lokalne definicje kolejek zdalnych mogą mieć różne priorytety domyślne z kolejek podstawowych, do których są one rozstrzygane. Jeśli w ścieżce rozstrzygania znajduje się więcej niż jedna definicja kolejki (patrz “Rozdzielczość nazwy” na stronie 835), domyślny priorytet jest przyjmowany z wartości (w momencie operacji put) atrybutu **DefPriority** kolejki określonej w komendzie open.

Wartość atrybutu **MaxPriority** menedżera kolejek to 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 między 0 (najniższy) i 9 (najwyższy).

Właściwości komunikatu

Za pomocą właściwości komunikatu można zezwolić aplikacji na wybór komunikatów do przetwarzania lub pobieranie informacji o komunikacie bez uzyskiwania dostępu do nagłówków MQMD lub MQRFH2 . Ułatwiają one także 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 na tematy lub do selektywnego pobierania komunikatów z kolejek. Właściwości komunikatu mogą być używane w celu uwzględnienia danych biznesowych lub informacji o stanie bez konieczności zapisywania ich w danych aplikacji. Aplikacje nie muszą uzyskiwać dostępu do danych w nagłówkach deskryptora komunikatu produktu MQ (MQMD) lub MQRFH2 , ponieważ pola w tych strukturach danych mogą być dostępne jako właściwości komunikatu przy użyciu wywołań funkcji interfejsu kolejki komunikatów (Message Queue Interface-MQI).

Użycie właściwości komunikatu w produkcie IBM MQ naśladuje użycie właściwości w produkcie JMS. Oznacza to, że można ustawiać właściwości w aplikacji JMS i pobierać je w proceduralnej aplikacji IBM MQ , a także w innej runze. Aby udostępnić właściwość aplikacji JMS , przypisz jej przedrostek "usr"; jest on wtedy dostępny (bez przedrostka) jako właściwość użytkownika komunikatu produktu JMS . For example, the IBM MQ property *usr.myproperty* (a character string) is accessible to a JMS application using the JMS call `message.getStringProperty('myproperty')` . Należy zauważyć, że aplikacje produktu JMS nie mogą uzyskać dostępu do właściwości z przedrostkiem "usr", jeśli zawierają one dwa lub więcej U+002E (".") . Właściwość bez przedrostka i nie ma wartości U+002E (".") Znak jest traktowany tak, jakby miał przedrostek "usr". I odwrotnie, można uzyskać dostęp do właściwości użytkownika ustawionej w aplikacji JMS w aplikacji IBM MQ , dodając "usr." w wywołaniu MQINQMP z przedrostkiem nazwy właściwości.

Właściwości komunikatu i długość komunikatu

Użyj atrybutu menedżera kolejek *MaxPropertiesLength* , aby kontrolować wielkość właściwości, które mogą przepływać przez dowolny komunikat w menedżerze kolejek produktu IBM MQ .

Ogólnie rzecz biorąc, jeśli właściwości MQSETMP są używane do ustawiania właściwości, wielkość właściwości to długość nazwy właściwości w bajtach, plus długość wartości właściwości w bajtach, która została przekazana do wywołania MQSETMP. Istnieje możliwość zmiany zestawu znaków nazwy właściwości i wartości właściwości podczas przesyłania komunikatu do miejsca docelowego, ponieważ mogą one zostać przekształcone w kod Unicode. W takim przypadku wielkość właściwości może ulec zmianie.

W wywołaniu MQPUT lub MQPUT1 właściwości komunikatu nie są wliczane do długości komunikatu dla kolejki i menedżera kolejek, ale są one wliczane do długości właściwości, które są postrzegane przez menedżer kolejek (niezależnie od tego, czy zostały one ustawione przy użyciu wywołań MQI właściwości komunikatu).

Jeśli wielkość właściwości przekracza maksymalną długość właściwości, komunikat jest odrzucany za pomocą komendy MQRC_PROPERTIES_TOO_BIG. Ponieważ wielkość właściwości jest zależna od jego reprezentacji, należy ustawić maksymalną długość właściwości na poziomie brutto.

Aplikacja może pomyślnie umieścić komunikat w buforze, który jest większy niż wartość parametru *MaxMsgLength*, jeśli bufor zawiera właściwości. Jest to spowodowane tym, że nawet jeśli są reprezentowane jako elementy MQRFH2 , właściwości komunikatu nie są wliczane do długości komunikatu. Pola nagłówka MQRFH2 są dodawane do długości właściwości tylko wtedy, gdy jeden lub więcej folderów jest zawartych, a każdy folder w nagłówku zawiera właściwości. Jeśli jeden lub więcej folderów znajduje się w nagłówku MQRFH2 , a dowolny folder nie zawiera właściwości, wówczas pola nagłówka MQRFH2 są liczone w kierunku długości komunikatu.

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

Nie należy używać zapytań o wartości *MaxMsgLength* , a następnie przydzielać bufor o takiej wielkości przed wywołaniem komendy MQGET. Zamiast tego należy przydzielić bufor, który jest wystarczająco duży. Jeśli wywołanie MQGET nie powiedzie się, należy przydzielić bufor z przewodnikiem o wielkości parametru *DataLength* .

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

Parametr *Buffer* wywołania MQPUT zawiera dane komunikatu aplikacji, które mają zostać wysłane, oraz wszystkie właściwości reprezentowane w danych komunikatu.

W przypadku przepływu do menedżera kolejek, który jest wcześniejszy niż IBM WebSphere MQ 7.0, właściwości komunikatu, z wyjątkiem tych, które znajdują się w deskryptorze komunikatu, są wliczane do długości komunikatu. Dlatego należy podnieść wartość atrybutu *MaxMsgLength* kanałów przechodnich do systemu w wersji wcześniejszej niż IBM WebSphere MQ 7.0 w razie potrzeby, aby zrekompensować fakt, że dla każdego komunikatu może być wysłanych więcej danych. Alternatywnie można obniżyć kolejkę lub menedżer kolejek *MaxMsgLength*, tak aby ogólny poziom przesyłanych danych w całym systemie pozostał taki sam.

Dla właściwości komunikatu istnieje limit długości 100 MB, wyłączając deskryptor komunikatu lub rozszerzenie dla każdego komunikatu.

Wielkość właściwości w wewnętrznej reprezentacji to długość nazwy, powiększona o jej wartość, a także niektóre dane sterujące dla właściwości. Istnieje również kilka danych sterujących dla zestawu właściwości po dodaniu jednej właściwości do komunikatu.

Nazwy właściwości

Nazwa właściwości to łańcuch znaków. Niektóre ograniczenia mają zastosowanie do jego długości i zestawu znaków, które mogą być używane.

Nazwa właściwości to łańcuch znaków, w którym rozróżniana jest wielkość liter, ograniczona do +4095 znaków, chyba że kontekst został ograniczony przez kontekst. Ten limit jest zawarty w stałej wartości MQ_MAX_PROPERTY_NAME_LENGTH.

W przypadku przekroczenia tej maksymalnej długości podczas korzystania z wywołania MQI właściwości komunikatu wywołanie kończy się niepowodzeniem z kodem przyczyny MQRC_PROPERTY_NAME_LENGTH_ERR.

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

W tym przypadku podczas analizy używane są tylko pierwsze 4095 znaków nazwy właściwości. Następujące znaki są obcinane. Może to spowodować, że aplikacja używała selektorów, aby nie były one zgodne z łańcuchem wyboru, lub aby były zgodne z łańcuchem, który nie jest oczekiwany, ponieważ więcej niż jedna właściwość może być obcięta do tej samej nazwy. Gdy nazwa właściwości zostanie obcięta, program WebSphereMQ wysyła komunikat dziennika błędów.

Wszystkie nazwy właściwości muszą być zgodne z regułami zdefiniowanymi przez specyfikację języka Java dla identyfikatorów Java , z tym wyjątkiem, że znak Unicode U+002E (.) jest dozwolony jako część nazwy-ale nie jest to początek. Reguły dla identyfikatorów Java są zgodne z regułami zawartymi w specyfikacji JMS dla nazw właściwości.

Znaki białych znaków i operatory porównania są zabronione. Osadzone wartości NULL są dozwolone w nazwie właściwości, ale nie są zalecane. W przypadku użycia wbudowanych wartości null zapobiega to stosowaniu stałej MQVS_NULL_TERMINATED, gdy jest ona używana z strukturą MQCHARV w celu określenia łańcuchów długości zmiennych.

Nazwy właściwości należy przechowywać w prosty sposób, ponieważ aplikacje mogą wybierać komunikaty w oparciu o nazwy właściwości, a konwersja między zestawem znaków nazwy i selektora może spowodować niespodziewanie przełączenie.

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

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

Dobrym sposobem uniknięcia starć nazw jest zapewnienie, aby wszystkie aplikacje prefiksował ich właściwości komunikatów z ich nazwą domeny internetowej. Na przykład w przypadku tworzenia aplikacji używającej nazwy domeny `ourcompany.com` można nazwać wszystkie właściwości przedrostkiem `com.ourcompany`. Ta konwencja nazewnictwa umożliwia także łatwy wybór właściwości. Na przykład aplikacja może uzyskać informacje o wszystkich właściwościach komunikatu, począwszy od `com.ourcompany.%`.

Więcej informacji na temat korzystania z nazw właściwości zawiera sekcja Ograniczenia dotyczące nazw właściwości .

Ograniczenia dotyczące nazw właściwości
Podczas tworzenia nazwy właściwości należy przestrzegać określonych reguł.

W przypadku nazw właściwości obowiązują następujące ograniczenia:

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

   Jedynymi wyjątkami są następujące właściwości udostępniające synonimy dla właściwości produktu JMS :

Właściwość	**Synonim dla**
JMSCorrelationID	Główny element.MQMD.CorrelId lub jms.Cid
JMSDeliveryMode	Główny element.MQMD.Persistence lub jms.Dlv
JMSDestination	jms.Dst
JMSExpiration	Główny element.MQMD.Expiry lub jms.Exp
JMSMessageID	Główny element.MQMD.MsgId
JMSPriority	Główny element.MQMD.Priority lub jms.Pri
JMSRedelivered	Główny element.MQMD.BackoutCount
JMSReplyTo (łańcuch zakodowany jako identyfikator URI)	Główny element.MQMD.ReplyToQ lub Root.MQMD.ReplyToQMgr lub jms.Rto
JMSTimestamp	Główny element.MQMD.PutDate lub Root.MQMD.PutTime lub jms.Tms
JMSType	mcd.Type lub mcd.Set lub mcd.Fmt
JMSXAppID	Główny element.MQMD.PutApplName

Właściwość	Synonim dla
JMSXDeliveryCount	Główny element.MQMD.BackoutCount
JMSXGroupID	Główny element.MQMD.GroupId lub jms.Gid
JMSXGroupSeq	Główny element.MQMD.MsgSeqNumber lub jms.Seq
JMSXUserID	Główny element.MQMD.UserIdentifier

Te synonimy umożliwiają aplikacji MQI uzyskanie dostępu do właściwości produktu JMS w sposób podobny do aplikacji klienckiej IBM MQ classes for JMS . Z tych właściwości można ustawić tylko atrybuty JMSCorrelationID, JMSReplyTo, JMSType, JMSXGroupIDi JMSXGroupSeq za pomocą interfejsu MQI.

Należy pamiętać, że właściwości JMS_IBM_ * dostępne w produkcie IBM MQ classes for JMS nie są dostępne przy użyciu interfejsu MQI. Pola, do których odwołują się odwołanie do właściwości JMS_IBM_ *, mogą być dostępne w inny sposób przez aplikacje MQI.

2. Właściwość nie może być wywoływana w żadnej mieszance mniejszej lub wielkiej litery, "NULL", "TRUE", "FALSE", "NOT", "AND", "OR", "BETWEEN", "LIKE", "IN", "IS" i "ESCAPE". Są to nazwy słów kluczowych języka SQL używanych w łańcuchach wyboru.
3. Początek nazwy właściwości " mq " w dowolnej mieszance pisanej małymi lub wielkimi literami, a nie zaczynając "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; nie można mieć pustego miejsca w hierarchii. Podobnie nazwa właściwości nie może kończyć się na "." na końcu.
5. Jeśli aplikacja ustawi właściwość "a.b", a następnie właściwość "a.b.c", nie jest jasne, czy w hierarchii "b" znajduje się wartość, czy też inna grupa logiczna. Taka hierarchia jest "mieszana treścią", a ta nie jest obsługiwana. Ustawienie właściwości, która powoduje, że treść mieszana nie jest dozwolona.

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

- Poprawność nazw właściwości jest sprawdzana podczas ustawiania właściwości przy użyciu wywołania MQSETMP-ustawianie właściwości komunikatu , jeśli sprawdzanie poprawności zostało zażądane podczas tworzenia uchwytu komunikatu. Jeśli podejmowana jest próba sprawdzenia poprawności właściwości i nie powiedzie się z powodu błędu w specyfikacji nazwy właściwości, kod zakończenia ma wartość MQCC_FAILED z powodu:
  - Błąd MQRC_PROPERTY_NAME_ERROR dla powodów 1-4.
  - MQRC_MIXED_CONTENT_NOT_ALLOWED dla przyczyny 5.
- Nazwy właściwości określonych bezpośrednio jako elementy MQRFH2 nie są gwarantowane w celu sprawdzenia poprawności 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*

Określ *Field* z tą samą sprawą co w przypadku pól struktury MQMD w deklaracji języka C. Na przykład nazwa właściwości `Root.MQMD.AccountingToken` uzyskuje dostęp do pola `AccountingToken` deskryptora komunikatu.

Pola `StrucId` i `Version` deskryptora komunikatu nie są dostępne z użyciem przedstawionej składni.

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

Jeśli dane komunikatu są uruchamiane za pomocą wywołania MQMDE, który został uhonorowany przez menedżer kolejek, można uzyskać dostęp do pól MQMDE za pomocą opisanego w nich 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

Właściwością może być wartość boolowska, łańcuch bajtowy, łańcuch znaków lub liczba zmiennopozycyjna lub liczba całkowita. Ta właściwość może przechowywać dowolną poprawną wartość w zakresie typu danych, chyba że kontekst został ograniczony 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, to znaczy jedną z wartością zerową.

Łańcuch bajtów nie jest poprawnym typem danych właściwości w produkcie 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 za pomocą komendy 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, które spełniają zapytanie SQL (Structured Query Language), które reprezentuje łańcuch wyboru.

Wybór przy użyciu wywołań funkcji MQSUB i MQOPEN

Za pomocą *`SelectionString`*, która jest strukturą typu MQCHARV, można dokonywać wyborów przy użyciu wywołań MQSUB i MQOPEN.

Struktura *`SelectionString`* jest używana do przekazywania łańcucha wyboru o zmiennej długości do menedżera kolejek.

Identyfikator CCSID powiązany z łańcuchem selektora jest ustawiany za pomocą pola VSCCSID w strukturze 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 ma obsługiwanej konwersji Unicode IBM MQ , powoduje wystąpienie błędu MQRC_SOURCE_CCSID_ERROR. Ten błąd jest zwracany w czasie, gdy selektor jest prezentowany w menedżerze kolejek, czyli w wywołaniu MQSUB, MQOPEN lub MQPUT1 .

Wartością domyślną w polu *VSCCSID* jest MQCCSI_APPL, co oznacza, że identyfikator CCSID łańcucha wyboru jest równy CCSID menedżera kolejek lub identyfikator CCSID klienta, jeśli jest on połączony

za pośrednictwem klienta. Stała MQCCSI_APPL może być jednak przesłonięta przez aplikację, która przedefiniowuje ją przed kompilacją.

Jeśli selektor MQCHARV reprezentuje łańcuch o wartości NULL, nie ma miejsca dla tego konsumenta komunikatów, a komunikaty są dostarczane tak, jakby selektor nie był używany.

Maksymalna długość łańcucha wyboru jest ograniczona tylko przez to, co może być opisane w polu MQCHARV *VSLength*.

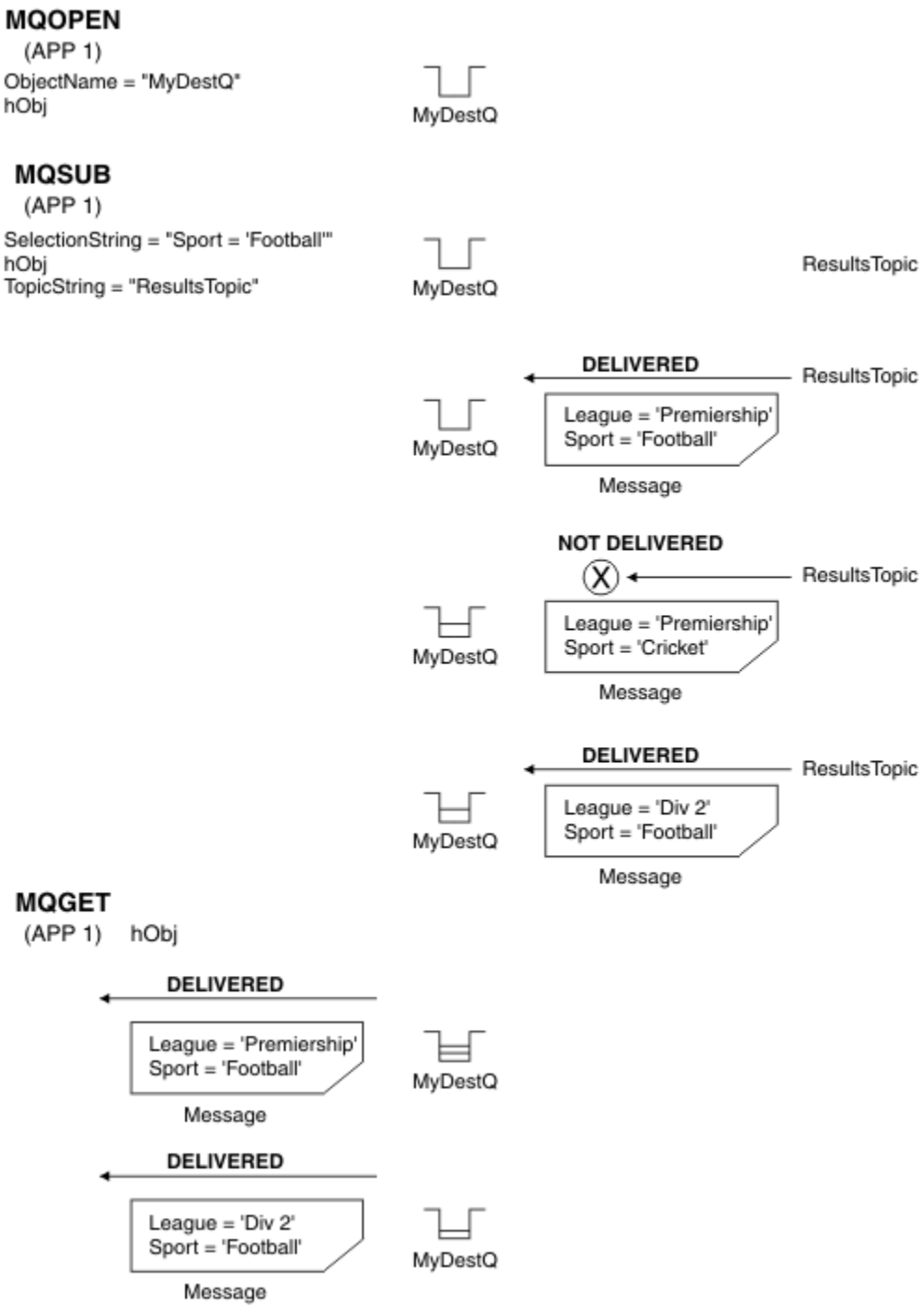
Element SelectionString jest zwracany w danych wyjściowych wywołania MQSUB przy użyciu opcji subskrypcji MQSO_RESUME, jeśli został podany bufor, a w polu VSBufSizeistnieje dodatnia długość buforu. Jeśli bufor nie zostanie podany, tylko długość łańcucha wyboru jest zwracana w polu długości VSLength tabeli MQCHARV. Jeśli podany bufor jest mniejszy niż obszar wymagany do zwrócenia pola, w udostępnionym buforze zwracane są tylko bajty VSBufSize .

Aplikacja nie może zmienić łańcucha wyboru bez najpierw zamknięcia uchwytu do kolejki (dla MQOPEN) lub subskrypcji (dla MQSUB). Nowy łańcuch wyboru może zostać następnie określony przy kolejnych wywołaniu MQOPEN lub MQSUB.

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

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

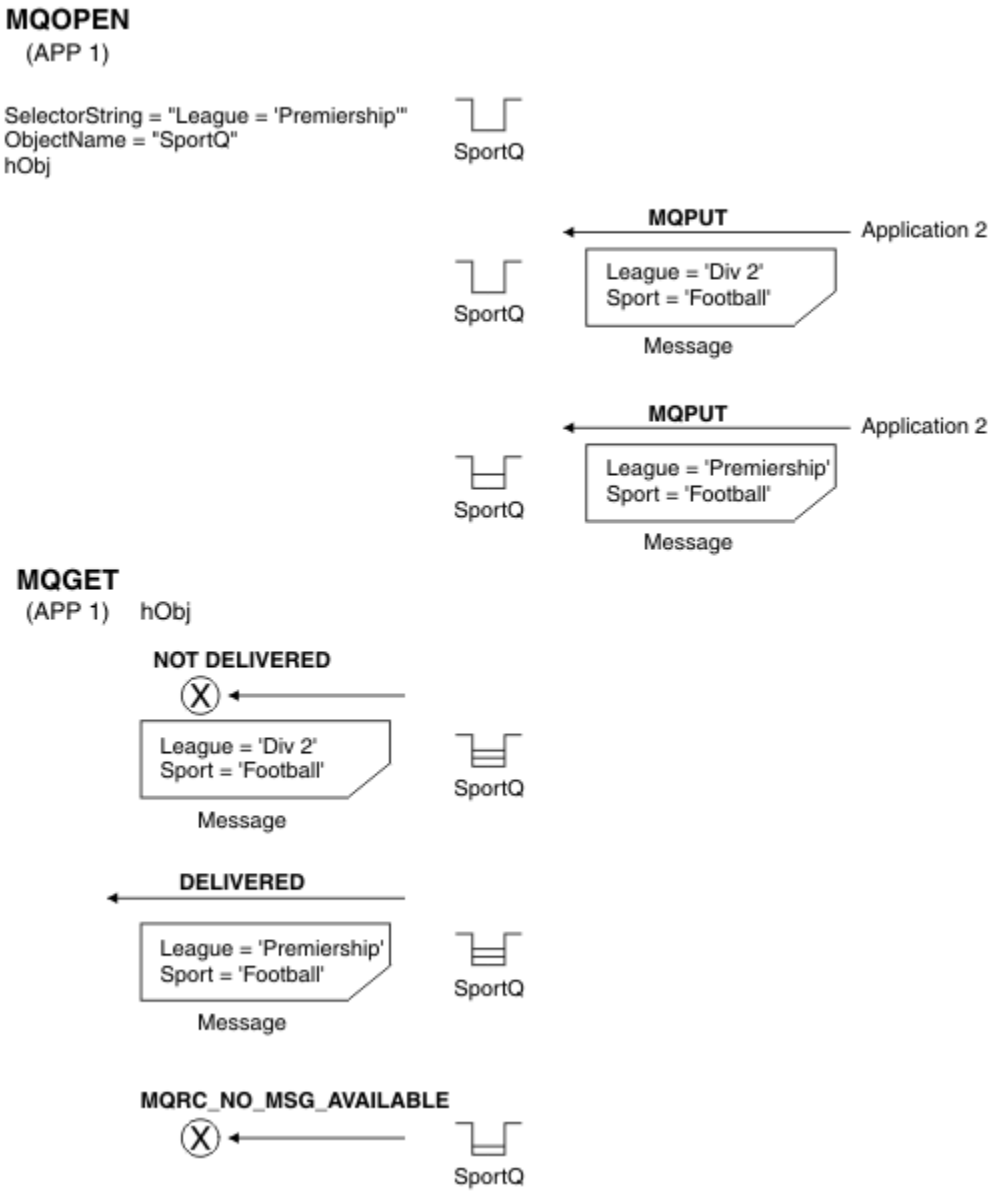
Rysunek 3 na stronie 32 przedstawia proces wyboru przy użyciu wywołania MQSUB.



Rysunek 3. Wybór przy użyciu wywołania MQSUB

Selektor można przekazać w wywołaniu do tabeli MQSUB, korzystając z pola *SelectionString* w strukturze MQSD. Efektem przekazania selektora w tabeli MQSUB jest to, że tylko te komunikaty publikowane w subskrybowanym temacie, które są zgodne z dostarczonym łańcuchem wyboru, są udostępniane w kolejce docelowej.

Rysunek 4 na stronie 33 przedstawia proces wyboru przy użyciu wywołania MQOPEN.



Rysunek 4. Wybór przy użyciu wywołania MQOPEN

Selektor może zostać przekazany w wywołaniu do wywołania MQOPEN za pomocą pola *SelectionString* w strukturze MQOD. Efektem przekazania selektora w wywołaniu MQOPEN jest to, że tylko te komunikaty w otwartej kolejce, które są zgodne z selektorem, są dostarczane do konsumenta komunikatów.

Głównym zastosowaniem selektora w wywołaniu MQOPEN jest przypadek punkt z punktem, w którym aplikacja może wybrać do odebrania 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 program MQOPEN, ale aplikacja pobierze tylko jeden z nich, ponieważ jest to jedyna taka aplikacja, która jest zgodna z selektorem.

Należy zauważyć, że kolejne wywołania MQGET w kolejce MQRC_NO_MSG_AVAILABLE nie istnieją w kolejce, która jest zgodna z podanym selektorem.

Pojęcia pokrewne

“Reguły i ograniczenia dotyczące łańcuchów wyboru” na stronie 40
Zapoznanie się z tymi regułami dotyczącymi sposobu interpretowania łańcuchów wyboru i ograniczenia znaków w celu uniknięcia potencjalnych problemów podczas korzystania z selektorów.

Zachowanie wyboru

Przegląd funkcji wyboru produktu IBM MQ .

Pola w strukturze MQMDE są uznawane za właściwości komunikatu dla odpowiednich właściwości deskryptora komunikatu, jeśli deskryptor MQMD:

- Ma format MQFMT_MD_EXTENSION
- Jest natychmiast po niej następuje poprawna struktura MQMDE
- Jest wersją jedną lub zawiera tylko dwa pola w wersji domyślnej.

Możliwe jest, aby łańcuch wyboru był rozstrzygany na wartość PRAWDA lub FAŁSZ, zanim zostaną wybrane wszystkie zgodne właściwości komunikatu. Na przykład może to być przypadek, gdy łańcuch wyboru jest ustawiony na wartość `"TRUE <> FALSE"`. Taka wczesna ocena jest gwarantowana, aby mieć miejsce tylko wtedy, gdy w łańcuchu wyboru nie ma odwołań do właściwości komunikatu.

Jeśli łańcuch wyboru zostanie rozstrzygany na wartość TRUE, zanim zostaną uwzględnione wszystkie właściwości komunikatu, zostaną dostarczone wszystkie komunikaty opublikowane w temacie zasubskrybowanym przez konsumenta. Jeśli łańcuch wyboru zostanie rozstrzygany na FALSE przed uznaniem wszystkich właściwości komunikatu, w wywołaniu funkcji, która zaprezentowała selektor, zostanie zwrócony kod przyczyny MQRC_SELECTOR_ALWAYS_FALSE, a kod zakończenia MQCC_FAILED.

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

Na przykład komunikat może nadal spełniać kryteria wyszukiwania, takie jak `'Color IS NULL'`, gdzie `'Color'` nie istnieje jako właściwość komunikatu w komunikacie.

Wybór może być wykonywany tylko dla właściwości powiązanych z komunikatem, a nie samego komunikatu, chyba że dostępny jest rozszerzony dostawca wyboru komunikatów. Wybór może być wykonywany na ładunku komunikatu tylko wtedy, gdy dostępny jest rozszerzony dostawca wyboru komunikatów.

Z każdą właściwością komunikatu jest powiązany typ. Podczas wykonywania wyboru należy upewnić się, że wartości używane w wyrażeniach do testowania właściwości komunikatu mają poprawny typ. Jeśli wystąpi niezgodność typów, wyrażenie, o którym mowa, jest tłumaczone na wartość FALSE.

Należy upewnić się, że łańcuch wyboru i właściwości komunikatu korzystają z zgodnych typów.

Kryteria wyboru są nadal stosowane w imieniu nieaktywnych trwałych subskrybentów, tak więc przechowywane są tylko komunikaty zgodne z łańcuchem wyboru, który został dostarczony.

Łańcuchy wyboru nie zmieniają się, gdy trwała subskrypcja jest wznawiana za pomocą komendy alter (MQSO_ALTER). Jeśli podczas wznawiania działania trwałego subskrybenta zostanie przedstawiony 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 spełniający kryteria wyboru.

Jeśli aplikacja określiła łańcuch wyboru zawierający wartości właściwości, wówczas tylko te komunikaty, które zawierają zgodne właściwości, są zakwalifikowane do wyboru. Na przykład subskrybent określa łańcuch wyboru "a = 3", a publikowany jest komunikat, który nie zawiera właściwości, lub właściwości, w których wartość 'a' nie istnieje lub nie jest równa 3. Subskrybent nie odbiera tego komunikatu do kolejki docelowej.

Wydajność przesyłania komunikatów

Wybranie komunikatów z kolejki wymaga, aby program IBM MQ sekwencyjnie inspekował każdy komunikat w kolejce. Komunikaty są sprawdzane do momentu znalezienia komunikatu, który jest zgodny z kryteriami wyboru lub nie ma więcej komunikatów do sprawdzenia. W związku z tym wydajność przesyłania komunikatów jest wystarczająca, jeśli wybór komunikatów jest używany w głębokich kolejkach.

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

- JMSCorrelationID = 'Identyfikator:*correlation_id*'
- JMSMessageID= 'ID:*id_komunikatu*'

gdzie:

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

Uwaga: Selektor powinien odwoływać się tylko do jednej z właściwości. Selektor, który ma jeden z tych formatów, zapewnia znaczącą poprawę wydajności podczas wybierania elementu JMSCorrelationID i oferuje marginalną poprawę wydajności dla JMSMessageID. Więcej informacji na ten temat zawiera "Selektory komunikatów w produkcie JMS" na stronie 138.

Korzystanie z złożonych selektorów

Selektory mogą zawierać wiele komponentów, na przykład:

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

Użycie takich złożonych selektorów może mieć poważne konsekwencje w zakresie wydajności i spowodować nadmierne zapotrzebowanie na zasoby. W związku z tym produkt IBM MQ będzie zabezpieczał system przez niepowodzenie przetwarzania nadmiernie złożonych selektorów, które mogą spowodować niedobór zasobów systemowych. Ochrona może wystąpić w przypadku łańcuchów wyboru, które zawierają więcej niż 100 testów lub gdy program IBM MQ wykryje, że zbliża się limit wielkości stosu systemu operacyjnego. Należy dokładnie spróbować i przetestować użycie łańcuchów wyboru z wieloma komponentami, na odpowiednich platformach, aby zapewnić, że limity ochrony nie są osiągane.

Wydajność i złożoność selektorów można poprawić, upraszczając je przy użyciu dodatkowego nawiasu łączącego komponenty. 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 40
Zapoznanie się z tymi regułami dotyczącymi sposobu interpretowania łańcuchów wyboru i ograniczenia znaków w celu uniknięcia potencjalnych problemów podczas korzystania z selektorów.

Składnia selektora komunikatów

Selektor komunikatów produktu IBM MQ jest łańcuchem składniowym, który jest oparty na podzbiorze składni wyrażenia warunkowego SQL92 .

Kolejność, w jakiej selektor komunikatów jest wartościowany, jest od lewej do prawej w obrębie poziomu priorytetu. Aby zmienić to zamówienie, można użyć nawiasów. Predefiniowane literały selektora i nazwy operatorów są zapisywane w tym miejscu wielkimi literami, jednak nie są one w nich rozróżniane.

Jeśli selektor jest udostępniany za pośrednictwem interfejsu API, program IBM MQ weryfikuje poprawność składową selektora komunikatów w momencie jego prezentacji. Jeśli składnia łańcucha wyboru jest niepoprawna lub nazwa właściwości nie jest poprawna, a dostawca wyboru komunikatów rozszerzonych jest niedostępny, do aplikacji zostanie zwrócony łańcuch `MQRC_SELECTION_NOT_AVAILABLE` . Jeśli składnia łańcucha wyboru jest niepoprawna lub nazwa właściwości nie jest poprawna, gdy subskrypcja jest wznawiana, do aplikacji jest zwracany `MQRC_SELECTOR_SYNTAX_ERROR` . Jeśli sprawdzanie poprawności nazwy właściwości zostało wyłączone podczas ustawiania właściwości (przez ustawienie `MQCMHO_NONE` zamiast `MQCMHO_VALIDATE`), a następnie aplikacja umieszcza komunikat z niepoprawną nazwą właściwości, ten komunikat nigdy nie zostanie wybrany.

Nie jest zwracany żaden błąd podczas prezentacji selektora, jeśli program IBM MQ określi, że selektor subskrypcji zdefiniowany administracyjnie korzysta z rozszerzonej składni komunikatów, zgodnie ze

wskazaniem parametru **DISPLAY SUB SELTYPE** o wartości EXTENDED. W takim przypadku sprawdzanie składni łańcucha wyboru jest odroczone do czasu publikacji (patrz MQRC_SELECTION_NOT_AVAILABLE).

Selektor może zawierać:

- Literały:
  - Literały łańcuchowe są ujęte w pojedynczy cudzysłów. Dwa kolejne apostrofowe znaki cudzysłowu reprezentują pojedynczy cudzysłów. Przykładami są literały i literał. Podobnie jak w przypadku literałów łańcuchowych Java , używane są kodowanie znaków Unicode. Nie można używać podwójnych cudzysłowów, aby ująć literał łańcuchowy. Każda sekwencja bajtów może być używana między pojedynczymi znakami cudzysłowu.
  - Łańcuch bajtowy to co najmniej jedna para znaków szesnastkowych ujęta w podwójny cudzysłów i poprzedzona przedrostkiem 0x. Przykłady: "0x2F1C" lub "0XD43A". Długość łańcucha bajtów musi wynosić co najmniej jeden bajt. Jeśli łańcuch bajtowy selektora jest dopasowany do właściwości komunikatu typu MQTYPE_BYTE_STRING, żadne działanie specjalne nie jest podejmowane w przypadku zera wiodącego lub końcowego. Bajty są traktowane jako inny znak. Nie rozważa się również endianness. Długość łańcuchów bajtów selektora i właściwości musi być taka sama, a sekwencja bajtów musi być taka sama.

    Przykłady wyboru łańcuchów bajtów (przyjęto, że *myBytes* = 0AFC23), które są zgodne:

    - "myBytes = "0x0AFC23"" = TRUE

    Następujące wybory łańcuchów 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 wiodące zero jest istotne w porównaniu)
    - "myBytes = "0x23FC0A"" = FALSE (ponieważ nie jest brane pod uwagę endianness)
  - Liczby szesnastkowe zaczynają się od zera, po którym następują wielkie lub małe litery x. Pozostała część literału zawiera jeden lub więcej poprawnych znaków szesnastkowych. Przykłady: 0xA, 0xAF, 0X2020.
  - Wiodąca wartość zero, po której następuje jedna lub więcej cyfr z zakresu 0-7, jest zawsze interpretowana jako początek liczby ósemkowej. Nie można reprezentować liczby dziesiętnej z przedrostkiem zero, takiej jak ta, na przykład, 09 zwraca błąd składniowy, ponieważ 9 nie jest poprawną cyfrą ósemkową. Przykłady liczb ósemkowych to 0177, 0713.
  - Dokładny literał liczbowy jest wartością liczbową bez przecinka dziesiętnego, np. 57, -957i +62. Dokładna literał liczbowy może zawierać wielkie lub małe litery L; nie ma to wpływu na sposób przechowywania lub interpretowania liczby. Produkt 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 zapisie naukowym, takim jak 7E3 lub -57.9E2, lub wartością liczbową z dziesiętnym, takim jak 7., -95.7lub +6.2. IBM MQ obsługuje liczby z zakresu od -1.797693134862315E+308 do 1.797693134862315E+308.

    Znacząca wartość powinna być zgodna z opcjonalnym znakiem (+ lub -). Wartość ta powinna być liczbą całkowitą lub ułamkową. Ułamkowa część znaczeń nie musi mieć wiodącej cyfry.

    Wielkie lub małe litery E wskazują na początek opcjonalnego wykładnika. Wykładnik ma wartość dziesiętną, a część wykładnika może być poprzedzona opcjonalnym znakiem znaku.

    Przybliżone literały liczbowe mogą zostać zakończone przez znak F lub D (bez rozróżniania wielkości liter). Ta składnia istnieje w celu obsługi metody języka znaczników o numerach pojedynczych lub podwójnych precyzji. Te znaki są opcjonalne i nie mają wpływu na sposób przechowywania lub przetwarzania literału liczbowego przybliżonego. Liczby te są zawsze zapisywane i przetwarzane za pomocą podwójnej precyzji.
  - Literały boolowskie TRUE i FALSE.

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

- Identyfikatory:

  Identyfikator jest sekwencją znaków o zmiennej długości, która musi rozpoczynać się od poprawnego znaku początkowego identyfikatora, po którym następuje zero lub więcej poprawnych znaków części identyfikatora. Reguły dla nazw identyfikatorów są takie same jak reguły dla nazw właściwości komunikatów, patrz "Nazwy właściwości" na stronie 27 i "Ograniczenia dotyczące nazw właściwości" na stronie 28 , aby uzyskać więcej informacji.

  Uwaga: Wybór może być wykonywany na ładunku komunikatu tylko wtedy, gdy dostępny jest rozszerzony dostawca wyboru komunikatów.

  Identyfikatory są odwołaniami do pól nagłówka lub odwołaniami do właściwości. Typ wartości właściwości w selektorze komunikatów musi odpowiadać typowi użytej do ustawienia właściwości, chociaż w miarę możliwości wykonywana jest promocja numeryczna. Jeśli wystąpi niezgodność typów, wynikiem wyrażenia jest `FALSE`. Jeśli przywoływana jest właściwość, która nie istnieje w komunikacie, jej wartością jest `NULL`.

  Konwersje typów, które mają zastosowanie do metod pobierania dla właściwości, nie mają zastosowania, jeśli właściwość jest używana w wyrażeniu selektora komunikatów. Jeśli na przykład właściwość zostanie ustawiona jako wartość łańcuchowa, a następnie zostanie użyta selektor w celu odpytania zapytania jako wartości liczbowej, wyrażenie zwróci wartość `FALSE`.

  JMS nazwy pól i właściwości, które są odwzorowywać na nazwy właściwości lub nazwy pól `MQMD` , są również poprawnymi identyfikatorami w łańcuchu wyboru. Produkt IBM MQ odwzorowuje rozpoznane pola JMS i nazwy właściwości na wartości właściwości komunikatu. Więcej informacji zawiera sekcja "Selektory komunikatów w produkcie JMS" na stronie 138. Jako przykład, łańcuch wyboru `"JMSPriority >="` wybiera właściwość `Pri` znalezionej w folderze `jms` bieżącego komunikatu.

- Przepełnienie/niedomiaru:

  Dla liczb dziesiętnych i przybliżonych liczb numerycznych nie są zdefiniowane następujące warunki:

  - Określanie liczby, która jest poza zdefiniowanym zakresem
  - Określanie wyrażenia arytmetycznego, które spowodowałoby przepełnienie lub niedomik.

  Dla tych warunków nie są wykonywane żadne sprawdzenia.

- Białe znaki:

  Zdefiniowana jako spacja, znak nowego wiersza, znak nowego wiersza, znak powrotu karetki, karta pozioma lub pionowa karta. 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 wartościowany do prawdziwych dopasowań; selektor, którego wartościowanie ma wartość false lub nieznany, nie jest zgodny.
  - Wyrażenia arytmetyczne składają się z samych siebie, operacji arytmetycznych, identyfikatorów (wartość identyfikatora jest traktowana jako literał liczbowy) oraz literałów liczbowych.
  - Wyrażenia warunkowe składają się z samych siebie, operacji porównania i operacji logicznych.
- Standard bracketing (), aby ustawić kolejność, w jakiej wyrażenia są wartościowane, jest obsługiwane.
- Operatory logiczne w kolejności kolejności wykonywania: `NOT`, `AND`, `OR`.
- Operatory porównania: =, >, >=, <, <=, <> (nie są równe).
  - Dwa łańcuchy bajtów są równe tylko wtedy, gdy łańcuchy mają taką samą długość, a sekwencja bajtów jest równa.
  - Porównywane są tylko wartości tego samego typu. Jedynym wyjątkiem jest to, że poprawne jest porównanie dokładnych wartości liczbowych i przybliżonych wartości liczbowych, (wymagana konwersja typów jest zdefiniowana przez reguły promocji liczbowej produktu Java). Jeśli istnieje próba porównania różnych typów, selektor zawsze ma wartość false.
  - Porównywanie łańcuchowe i boolowskie jest ograniczone do produktów = i <>. Dwa łańcuchy są równe tylko wtedy, gdy zawierają taką samą sekwencję znaków.
- Operatory arytmetyczne w kolejności kolejności wykonywania zadań:
  - +, - unary.
  - * mnożenie i / .
  - + oraz - odejmowanie.
  - Operacje arytmetyczne na wartości NULL nie są obsługiwane. Jeśli są one uruchomione, pełny selektor zawsze ma wartość false.
  - Operacje arytmetyczne muszą używać promocji numerycznej 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 dowolna z wyrażeń operacji `BETWEEN` ma wartość NULL, wartością tej operacji jest false (fałsz). Jeśli dowolna z wyrażeń operacji `NOT BETWEEN` to `NULL`, wartość tej operacji jest prawdziwa.
- identyfikator `[NOT] IN (string-literal1, string-literal2,...)` operator porównania, gdzie identyfikator ma wartość typu String lub `NULL` .
  - Wartość `Country IN ('UK', 'US', 'France')` ma wartość true dla `'UK'` i false dla `'Peru'`. Jest on równoważny z wyrażeniem `(Country = 'UK') OR (Country = 'US') OR (Country = 'France')`.
  - `Country NOT IN ('UK', 'US', 'France')` ma wartość false dla `'UK'` i ma wartość true dla `'Peru'`. Jest on równoważny z wyrażeniem `NOT ((Country = 'UK') OR (Country = 'US') OR (Country = 'France'))`.
  - Jeśli identyfikator operacji `IN` lub `NOT IN` ma wartość NULL, wartość tej operacji jest nieznana.
- `identifier [NOT] LIKE` *`pattern-value`* `[ESCAPE` *`escape-character`* `]` , operator porównania, gdzie `identifier` ma wartość łańcuchową. *wartość-wzorca* jest literałem łańcuchowym, gdzie _ oznacza dowolny pojedynczy znak, a % oznacza dowolną sekwencję znaków (włącznie z pustą sekwencją). Wszystkie inne postacie stoją dla siebie. Opcjonalny *znak zmiany znaczenia* to pojedynczy literał łańcuchowy, który jest używany do zmiany znaczenia specjalnego znaczenia _ i % w *wartości wzorca*. Operator `LIKE` musi być używany tylko do porównywania dwóch wartości łańcuchowych.
  - Wartość `phone LIKE '12%3'` ma wartość true dla 123 i 12993, a wartość false dla 1234.
  - `word LIKE 'l_se'` ma wartość true w przypadku utraty i wartości false dla poluzowania.

– Wartość `underscored LIKE '\_%' ESCAPE '\'` ma wartość true dla `_foo` i false dla `bar`.
– `phone NOT LIKE '12%3'` ma wartość false dla 123 oraz 12993 i ma wartość true dla 1234.
– Jeśli identyfikator operacji `LIKE` lub `NOT LIKE` ma wartość `NULL`, wartość tej operacji jest nieznana.

Uwaga: Operator `LIKE` musi być używany do porównywania dwóch wartości łańcuchowych. Wartość `Root.MQMD.CorrelId` to 24-bajtowa tablica bajtów, a nie łańcuch znaków. Łańcuch selektora `Root.MQMD.CorrelId LIKE 'ABC%'` jest akceptowany przez analizator składni jako poprawny pod względem składniowym, ale jest wartościowany do wartości false. W przypadku porównywania tablicy bajtów z łańcuchem znaków program `LIKE` nie może być w tym przypadku używany.

- Testy operatora porównania `identifier IS NULL` dla wartości pola nagłówka `NULL` lub brakującej wartości właściwości.
- Operator porównania `identifier IS NOT NULL` testuje istnienie wartości pola nagłówka o wartości innej niż NULL lub wartości właściwości.
- Wartości NULL

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

  – Język SQL traktuje wartość `NULL` jako nieznaną.
  – Porównanie lub arytmetyka z nieznaną wartością zawsze daje nieznaną wartość.
  – Operatory `IS NULL` i `IS NOT NULL` przekształcają nieznaną wartość w wartości `TRUE` i `FALSE` .

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

Tabela 1. Wartość boolowskiego wyniku operatora, gdy logika jest 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 boolowskiego wyniku operatora, gdy logika jest 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
U	T	T
U	U	U

Tabela 2. Wartość boolowskiego wyniku operatora, gdy logika jest A OR B (kontynuacja)		
Operator A	**Operator B**	**Wynik (A OR B)**
U	F	U

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

Poniższy selektor komunikatów wybiera komunikaty z typem komunikatu samochodu, koloru niebieskiego i wagą większą niż 2500 funtów:

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

Chociaż SQL obsługuje stałe porównania dziesiętne i arytmetyczne, selektory komunikatów nie są dostępne. Dlatego dokładne literały liczbowe są ograniczone do tych bez liczby dziesiętnej. Dlatego też istnieją cyfry z separatą dziesiętną jako alternatywna reprezentacja dla przybliżonej wartości liczbowej.

Komentarze SQL nie są obsługiwane.

Pojęcia pokrewne

“Właściwości komunikatu” na stronie 26
Za pomocą właściwości komunikatu można zezwolić aplikacji na wybór komunikatów do przetwarzania lub pobieranie informacji o komunikacie bez uzyskiwania dostępu do nagłówków MQMD lub MQRFH2 . Ułatwiają one także komunikację między aplikacjami IBM MQ i JMS .

Odsyłacze pokrewne

MsgHandle

MQBUFMH-przekształcanie buforu w uchwyt komunikatu

Reguły i ograniczenia dotyczące łańcuchów wyboru

Zapoznanie się z tymi regułami dotyczącymi sposobu interpretowania łańcuchów wyboru i ograniczenia znaków w celu uniknięcia potencjalnych problemów podczas korzystania z selektorów.

- Wybór komunikatów na potrzeby przesyłania komunikatów w trybie publikowania/subskrypcji odbywa się na komunikacie wysłanym przez publikatora. Patrz: łańcuchy wyboru.
- Równoważność jest testowana przy użyciu pojedynczego znaku równości, na przykład `a = b` jest poprawna, podczas gdy `a == b` jest niepoprawna.
- Operatorem używanym przez wiele języków programowania do reprezentowania "not equal to" jest `!=`. Ta reprezentacja nie jest poprawnym synonimem produktu <> ; Na przykład wartość `a <> b` jest poprawna, natomiast wartość `a != b` nie jest poprawna.
- Pojedyncze cudzysłowy są rozpoznawane tylko wtedy, gdy " (U+0027) jest używany znak. Podobnie, podwójne cudzysłowy, poprawne tylko wtedy, gdy są używane do ujmowania łańcuchów bajtowych, muszą używać znaku " (U+0022) znak.
- Symbole &, &&, | i | | nie są synonimami dla logicznego łączenia/rozłączenia; na przykład `a && b` musi być określony jako `a AND b`.
- Znaki wieloznaczne * i ? nie są synonimami dla produktów % i _.
- Selektory zawierające wyrażenia złożone, takie jak `20 < b < 30` , nie są poprawne. Analizator składni ocenia operatorów, którzy mają ten sam priorytet od lewej do prawej. W związku z tym 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 podwójny cudzysłów; jeśli używane są pojedyncze cudzysłowy, to łańcuch bajtowy jest przyjmowany jako literał łańcuchowy. Liczba znaków (nie liczba znaków reprezentowanych przez znaki) po `0x` musi być wielokrotnością dwóch znaków.
- Słowo kluczowe `IS` nie jest synonimem znaku równości. W ten sposób łańcuchy wyboru `a IS 3` i `b IS 'red'` nie są poprawne. Słowo kluczowe `IS` istnieje tylko w celu obsługi przypadków `IS NULL` i `IS NOT NULL`.

Pojęcia pokrewne

Łańcuchy wyboru

“Zachowanie wyboru” na stronie 34

Przegląd funkcji wyboru produktu IBM MQ .

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

Znaki, które nie są ujęte w pojedynczy cudzysłów, które tworzą zastrzeżone słowa kluczowe łańcucha wyboru, muszą zostać wprowadzone w języku Basic Latin Unicode (od znaku U+0000 do U+0007F). Użycie innych reprezentacji znaków alfanumerycznych nie jest poprawne. Na przykład liczba 1 musi być wyrażona jako U+0031 w kodzie Unicode, nie jest ona poprawna, aby używać ekwiwalentu Fullwidth Digit U+FF11 lub arabskiego odpowiednika U+0661.

Nazwy właściwości komunikatów mogą być określane przy użyciu dowolnej poprawnej sekwencji znaków Unicode. Nazwy właściwości komunikatów zawarte w łańcuchach wyboru, które są kodowane w UTF-8 , będą sprawdzane nawet wtedy, gdy zawierają znaki wielobajtowe. Sprawdzanie poprawności wielobajtowych UTF-8 jest ścisłe i należy upewnić się, że do nazw właściwości komunikatów używane są poprawne sekwencje UTF-8 . Znaki spoza języka Basic Multilingual Plane (powyższe U + FFFF), reprezentowane w formacie UTF-16 przez zastępcze punkty kodowe (X'D800'przez X'DFFF') lub cztery bajty w UTF-8, nie są obsługiwane w nazwach właściwości komunikatu.

Podczas porównywania pod kątem równości nie jest wykonywane żadne dodatkowe przetwarzanie w nazwach właściwości ani wartościach. Oznacza to na przykład, że nie ma miejsca przed/de-kompozycja i ligatury nie są podane w sposób szczególny. Na przykład wstępnie złożony znak umlaut U+00FC nie jest uznawany za równoważny U+0075 + U+0308 , a sekwencja znaków nie jest uznawana za 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 poprawność.

Wybieranie treści komunikatu

Istnieje możliwość zasubskrybowania na podstawie wyboru zawartości ładunku komunikatu (zwanej również filtrowaniem treści), ale decyzja o tym, które komunikaty powinny zostać dostarczone do takiej subskrypcji, nie może być wykonywana bezpośrednio przez produkt IBM MQ. Zamiast tego wymagany jest rozszerzony dostawca wyboru komunikatów, na przykład IBM Integration Bus, w celu przetworzenia komunikatów.

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

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

Jeśli rozszerzony dostawca wyboru komunikatów stwierdzi, że publikacja nie jest zgodna, komunikat nie zostanie dostarczony do subskrybenta. Może to spowodować, że wywołanie MQPUT lub MQPUT1 nie powiedzie się, a kod przyczyny MQRC_PUBLICATION_FAILURE. Jeśli dostawca wyboru rozszerzonego komunikatu nie może przeanalizować publikacji, zwracany jest kod przyczyny MQRC_CONTENT_ERROR, a wywołanie MQPUT lub MQPUT1 nie powiodło się.

Jeśli dostawca wyboru rozszerzonego komunikatu nie jest dostępny lub nie może określić, czy subskrybent powinien otrzymać publikację, zwracany jest kod przyczyny MQRC_SELECTION_NOT_AVAILABLE, a wywołanie MQPUT lub MQPUT1 nie powiedzie się.

Gdy subskrypcja jest tworzona z użyciem filtru treści, a dostawca wyboru rozszerzonego komunikatu nie jest dostępny, wywołanie MQSUB kończy się niepowodzeniem z kodem przyczyny MQRC_SELECTION_NOT_AVAILABLE. Jeśli subskrypcja z filtrem treści jest wznawiana, a dostawca wyboru komunikatów rozszerzonych jest niedostępny, wywołanie MQSUB zwróci ostrzeżenie o wartości MQRC_SELECTION_NOT_AVAILABLE, ale subskrypcja może zostać wznowiona.

Pojęcia pokrewne

Łańcuchy wyboru

Asynchroniczne wykorzystanie komunikatów produktu IBM MQ

Wykorzystanie asynchroniczne korzysta z zestawu rozszerzeń MQI (Message Queue Interface), wywołania MQI MQCB i MQCTL, które pozwalają na zapisywanie aplikacji MQI w celu odbierania komunikatów z zestawu kolejek. Komunikaty są dostarczane do aplikacji przez wywołanie 'jednostki kodu', identyfikowanej przez aplikację, przekazując komunikat lub leksem reprezentującym komunikat.

W najprostszych środowiskach aplikacji jednostka kodu jest definiowana za pomocą wskaźnika funkcji, jednak w innych środowiskach jednostka kodu może być zdefiniowana przez nazwę programu lub modułu.

W przypadku asynchronicznego korzystania z komunikatów używane są następujące terminy:

Konsument komunikatu
: Konstrukcja programistyczny, która umożliwia zdefiniowanie programu lub funkcji, które mają być wywoływane z komunikatem, gdy jest on zgodny z wymaganiami aplikacji.

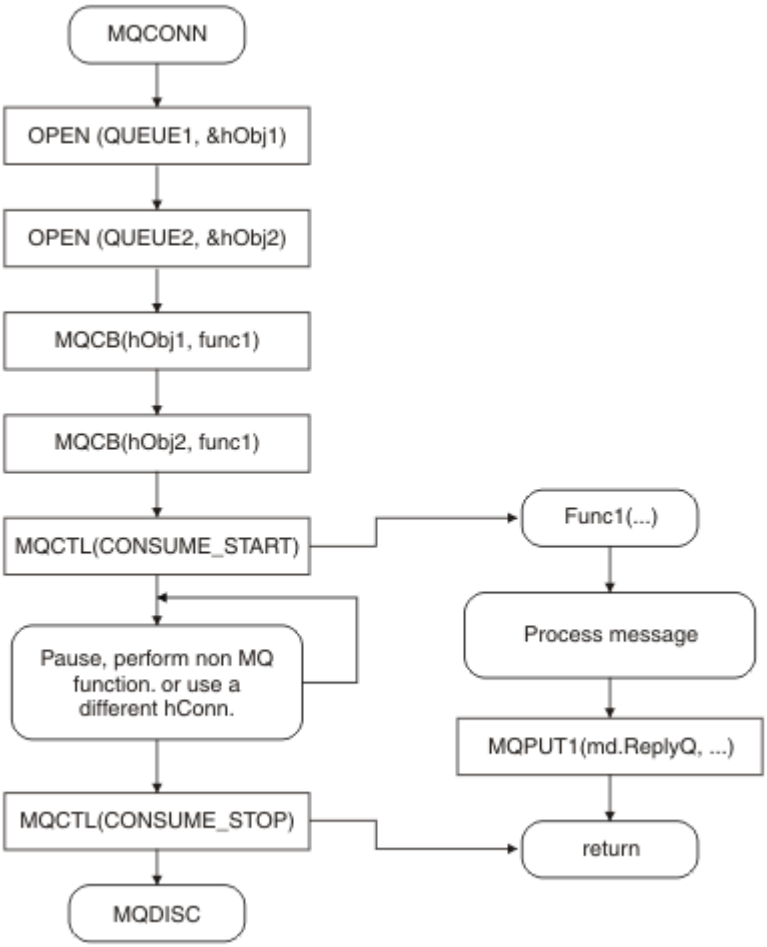
procedura obsługi zdarzeń
: Konstrukcja programistyczny, która umożliwia zdefiniowanie programu lub funkcji, która ma zostać wywołana, gdy wystąpi zdarzenie asynchroniczne, takie jak wygaszanie menedżera kolejek.

Wywołanie zwrotne
: Termin ogólny używany do odwołania się do procedury obsługi komunikatów lub do procedury obsługi zdarzeń.

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

Poniższe ilustracje dają przykład, w jaki sposób można użyć tej funkcji.

Program Rysunek 5 na stronie 43 wyświetla wielowątkową aplikację, która konsumuje komunikaty z dwóch kolejek. W przykładzie przedstawiono wszystkie komunikaty dostarczane do jednej funkcji.

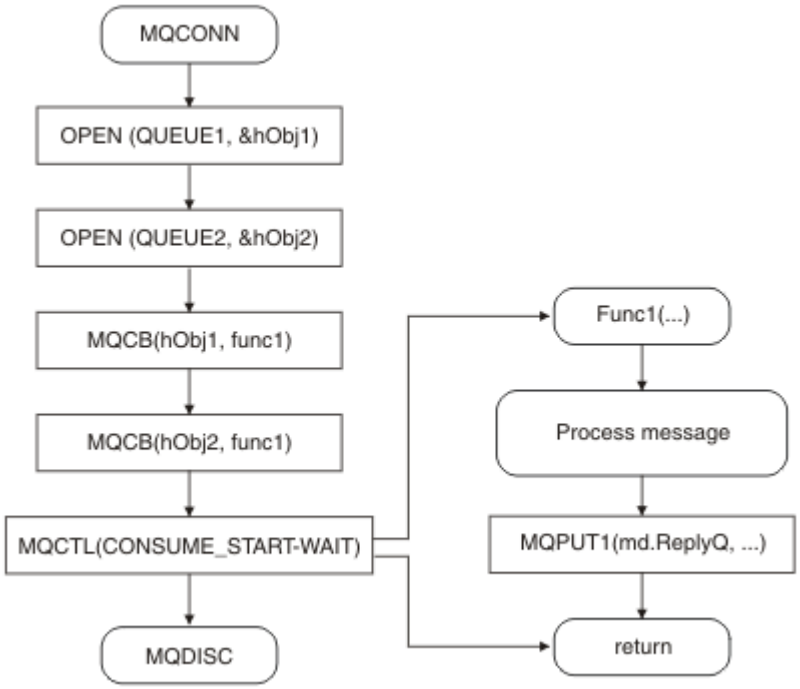


Rysunek 5. Standardowa aplikacja sterowana komunikatami wykorzystująca dwie kolejki

z/OS W systemie z/OSgłówny wątek sterujący musi wywołać wywołanie MQDISC przed zakończeniem. Umożliwia to dowolne wątki wywołania zwrotnego w celu zakończenia i zwalniania zasobów systemu.

Rysunek 6 na stronie 44 Ten przykładowy przepływ przedstawia jednowątkową aplikację, która konsumuje komunikaty z dwóch kolejek. W przykładzie przedstawiono wszystkie komunikaty dostarczane do jednej funkcji.

Różnica między przypadkiem asynchronicznym jest taka, że element sterujący nie wraca do wystawcy MQCTL, dopóki wszystkie konsumenty nie dezaktywują się. To oznacza, że jeden konsument wygenerował żądanie zatrzymania MQCTL lub wygaszanie menedżera kolejek.



Rysunek 6. Aplikacja jednowątkowa sterowana komunikatami korzystała z dwóch kolejek

Grupy komunikatów

Komunikaty mogą występować w grupach, aby zezwolić na porządkowanie komunikatów.

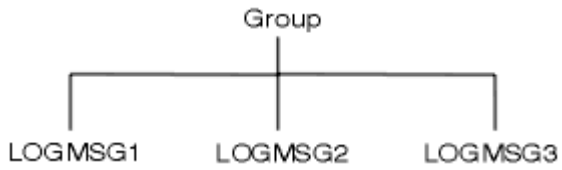
Grupy komunikatów umożliwiają oznaczanie wielu komunikatów jako powiązanych z nimi, a także kolejność logiczna, która ma być zastosowana do grupy (patrz "Uporządkowanie logiczne i fizyczne" na stronie 866). W systemie Wiele platform segmentacja komunikatów umożliwia rozbicie dużych komunikatów na mniejsze segmenty. podczas wprowadzania do tematu nie można używać zgrupowanych ani segmentowanych 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 *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 zwracanego przez pojedynczą operację MQGET, która nie określa parametru MQGMO_COMPLETE_MSG.

Rysunek 7 na stronie 44 przedstawia grupę komunikatów logicznych:



Rysunek 7. Grupa komunikatów logicznych

Otwarcie kolejki i określenie parametru MQOO_BIND_ON_GROUP powoduje wymuszenie wysłania wszystkich komunikatów w grupie, które są wysyłane do tej kolejki, do tej samej instancji kolejki. Więcej informacji na temat opcji BIND_ON_GROUP znajduje się w sekcji Obsługa powinowactw komunikatów.

Komunikat logiczny

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

Użyj komunikatów logicznych w grupie do:

- Upewnij się, że zamawiający (jeśli nie jest to gwarantowane w okolicznościach, w których wiadomość jest przesyłana).
- Zezwalaj aplikacjom na grupowanie podobnych komunikatów (na przykład te, które muszą być przetwarzane 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 logicznie odrębnym komunikatem, a tylko pola *GroupId* i *MsgSeqNumber* w strukturze MQMD muszą zawierać dowolne relacje z innymi komunikatami w grupie. Inne pola w strukturze MQMD są niezależne; niektóre z nich 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 kodowane.

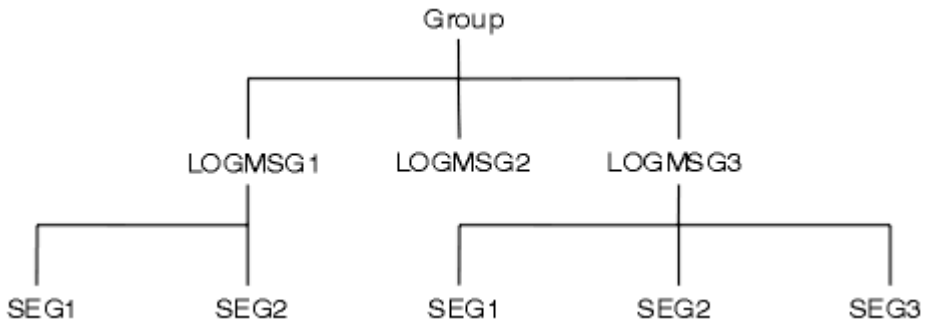
Segment

Segmenty są używane do obsługi komunikatów, które są zbyt duże dla umieszczania lub pobierania aplikacji lub menedżera kolejek (w tym między tymi menedżerami kolejek, przez które przekazywane są komunikaty). Więcej informacji na ten temat zawiera sekcja “Segmentacja komunikatów” na stronie 884.

Pojedynczy komunikat jest podzielony na mniejsze komunikaty o nazwie *segmenty*. Segment komunikatu jest identyfikowany przez pola *GroupId*, *MsgSeqNumber*i *Offset* . Pole *Offset* zaczyna 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 45 przedstawia przykład komunikatów w grupie). Segment jest logicznie częścią pojedynczego komunikatu, dlatego tylko pola *MsgId*, *Offset*i *MsgFlags* w strukturze MQMD powinny różnić się między oddzielnymi segmentami tego samego komunikatu. Jeśli nie uda się dojść do segmentu, kod przyczyny MQRC_INCOMPLETE_GROUP lub MQRC_INCOMPLETE_MSG jest zwracany jako odpowiedni.

W programie Rysunek 8 na stronie 45 wyświetlana jest grupa komunikatów logicznych, z których część jest segmentowana:



Rysunek 8. Posegmentowane komunikaty

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

Nie można używać segmentowanych lub zgrupowanych komunikatów z publikowania/subskrybowania.

Pojęcia pokrewne

“Segmentacja komunikatów” na stronie 884
Ta sekcja zawiera informacje na temat segmentacji komunikatów. Ta funkcja nie jest obsługiwana w produkcie IBM MQ for z/OS ani przez aplikacje korzystające z produktu IBM MQ classes for JMS.

Odsyłacze pokrewne

“Uporządkowanie logiczne i fizyczne” na stronie 866
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 jest restartowany po awarii, odtwarza te trwałe komunikaty w miarę potrzeb od zarejestrowanych danych. Komunikaty, które nie są trwałe, są usuwane w przypadku zatrzymania menedżera kolejek, bez względu na to, czy zatrzymanie jest wynikiem komendy operatora, czy też z powodu awarii pewnej części systemu.

z/OS Nietrwałe komunikaty zapisane w narzędziu CF (Coupling Facility) w systemie z/OS są dla tego wyjątku wyjątkiem. Utrzymują się tak długo, jak długo system CF pozostaje dostępny.

Jeśli podczas tworzenia komunikatu zainicjowano deskryptor komunikatu (MQMD) przy użyciu wartości domyślnych, trwałość komunikatu jest pobierana z atrybutu **DefPersistence** kolejki określonego 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 komunikaty trwałe; zakres działania zależy od charakterystyki wydajności podsystemu we/wy komputera oraz sposobu korzystania z opcji punktu synchronizacji na każdej platformie:

- Trwały komunikat, poza bieżącą jednostką pracy, jest zapisywany na dysku przy każdym operacji umieszczania i pobierania. Patrz “Zatwierdzanie i wycofywanie jednostek pracy” na stronie 946.
- ULW z/OS W przypadku wszystkich platform z wyjątkiem produktu IBM ikomunikat trwały w bieżącej jednostce pracy jest rejestrowany tylko wtedy, gdy jednostka pracy jest zatwierdzana, a jednostka pracy może zawierać wiele operacji w kolejce.

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

Uwaga: Kombinacja zapisu trwałych komunikatów w jednostce pracy oraz zapisywania trwałych komunikatów poza jednostką lub pracą może powodować potencjalnie poważne problemy z wydajnością aplikacji. Jest to szczególnie prawdziwe, gdy dla obu operacji używana jest ta sama kolejka docelowa.

Komunikaty, których dostarczenie nie powiodło się

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

Można wykonać następujące czynności:

- 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 proceduralnych błędów programu” na stronie 1145.

Komunikaty, których kopie zapasowe zostały wycofane

W przypadku przetwarzania komunikatów z kolejki pod kontrolą jednostki pracy, jednostka pracy może składać się z jednego lub większej liczby komunikatów. W przypadku wystąpienia wycofania komunikaty, które zostały pobrane z kolejki, zostają przywrócone w kolejce i mogą być ponownie przetwarzane w innej jednostce pracy. Jeśli przetwarzanie konkretnego komunikatu jest przyczyną problemu, kopia zapasowa jednostki pracy jest wycofana ponownie. Może to spowodować pętlę przetwarzania. Komunikaty, które zostały umieszczone w kolejce, są usuwane z kolejki.

Aplikacja może wykrywać komunikaty, które są wychwytywać w takiej pętli, testując pole *BackoutCount* deskryptora MQMD. Aplikacja może poprawić sytuację lub wywołać ostrzeżenie dla operatora.

Liczba wycofań zawsze jest restartowana restartami menedżera kolejek. Każda zmiana atrybutu **HardenGetBackout** jest ignorowana.

W przypadku kolejek współużytkowanych licznik wycofań zawsze jest restartowany restartami menedżera kolejek. W przypadku wszystkich innych konfiguracji w systemie z/OS, aby upewnić się, że liczba wycofań dla kolejek prywatnych jest restartowana od menedżera kolejek, należy ustawić atrybut *HardenGetBackout* na wartość MQQA_BACKOUT_HARDENED; w przeciwnym razie, jeśli menedżer kolejek ma zostać zrestartowany, nie utrzymuje dokładnej liczby wycofań dla każdego komunikatu. Ustawienie atrybutu w ten sposób powoduje dodanie kosztu dodatkowego przetwarzania.

Więcej informacji na temat zatwierdzania i tworzenia kopii zapasowych komunikatów zawiera sekcja “Zatwierdzanie i wycofywanie jednostek pracy” na stronie 946.

Kolejka zwrotna i menedżer kolejek

Mogą wystąpić sytuacje, w których można odbierać komunikaty w odpowiedzi na komunikat, który jest wysyłany:

- Komunikat odpowiedzi w odpowiedzi na komunikat żądania
- Komunikat raportu na temat nieoczekiwanego zdarzenia lub utraty ważności
- Komunikat raportu o zdarzeniu COA (Potwierdzenie przyjazdu) lub COD (Potwierdzenie dostarczenia).
- Komunikat raportu o zdarzeniu PAN (Positive Action Notification) lub NAN (Negative Action Notification).

Przy użyciu struktury MQMD należy określić nazwę kolejki, do której mają być wysyłane komunikaty odpowiedzi i raporty w polu *ReplyToQ* . W polu *ReplyToQMgr* podaj nazwę menedżera kolejek, do którego należy kolejka odpowiedzi.

Jeśli pole *ReplyToQMgr* pozostanie puste, menedżer kolejek ustawia zawartość 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 pole to nie jest zmieniane.

ReplyToQMgr
Jeśli *ReplyToQ* jest lokalną definicją kolejki zdalnej, pole *ReplyToQMgr* jest ustawione na nazwę menedżera kolejek, do którego należy kolejka zdalna. W przeciwnym razie pole *ReplyToQMgr* jest ustawione na nazwę menedżera kolejek, z którym połączona jest aplikacja.

Uwaga: Można zażądać, aby menedżer kolejek udostępnił więcej niż jedną próbę dostarczenia komunikatu, a następnie można zażądać, aby komunikat został odrzucony, jeśli nie powiedzie się. Jeśli komunikat, po niepowodzeniu dostarczenia, nie zostanie usunięty, zdalny menedżer kolejek umieszcza komunikat w swojej kolejce niedostarczonych komunikatów (komunikat niedostarczonych komunikatów) (patrz sekcja “Korzystanie z kolejki niedostarczonych komunikatów (niedostarczonych komunikatów)” na stronie 1148).

Kontekst komunikatu

Informacja *Kontekst komunikatu* umożliwia aplikacji, która pobiera komunikat, w celu uzyskania informacji o inicjatorze komunikatu.

Pobieranie aplikacji może być następujące:

- Sprawdź, czy aplikacja wysyłający ma właściwy poziom uprawnień
- Wykonaj niektóre funkcje księgowe, aby móc pobierać aplikację wysyłający dla każdej pracy, którą musi wykonać.
- Przechowuj zapis kontrolny wszystkich komunikatów, z którymi współpracował

W przypadku użycia wywołania MQPUT lub MQPUT1 w celu umieszczenia komunikatu w kolejce można określić, że menedżer kolejek ma dodać pewne domyślne informacje kontekstu do deskryptora komunikatu. Aplikacje, które mają odpowiedni poziom uprawnień, mogą dodawać dodatkowe informacje kontekstowe. Więcej informacji na temat sposobu określania informacji o kontekście zawiera sekcja “Sterowanie informacjami o kontekście komunikatu” na stronie 851.

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

- Potwierdź przy dostarczeniu
- Utrata ważności

Gdy te komunikaty są generowane, kontekst użytkownika jest sprawdzany pod kątem uprawnień + put i + passid na miejscu docelowym raportu. Jeśli kontekst użytkownika ma niewystarczające uprawnienia, komunikat raportu jest umieszczany w kolejce niedostarczonych komunikatów, jeśli został on zdefiniowany. Jeśli nie ma kolejki niedostarczonych komunikatów, komunikat raportu jest odrzucany.

Wszystkie informacje kontekstowe są zapisywane w polach kontekstu deskryptora komunikatu. Typ informacji jest objęty tożsamością, pochodzeniem i informacjami o kontekście użytkownika.

kontekst tożsamości

Kontekst tożsamości -informacje identyfikują użytkownika aplikacji, która po raz pierwszy umiełała komunikat w kolejce. Odpowiednio autoryzowane aplikacje mogą ustawiać następujące pola:

- Menedżer kolejek wypełnia pole *UserIdentifier* nazwą identyfikującą użytkownika. Sposób działania menedżera kolejek zależy od środowiska, w którym aplikacja jest uruchomiona.
- Menedżer kolejek wypełnia pole *AccountingToken* znacznikiem lub numerem określonym w aplikacji, w której znajduje się komunikat.
- Aplikacje mogą korzystać z pola *ApplIdentityData* w celu uzyskania dodatkowych informacji, które mają zostać dołączone do użytkownika (na przykład zaszyfrowane hasło).

Identyfikator zabezpieczeń systemu Windows (SID) jest przechowywany w polu *AccountingToken* , gdy komunikat jest tworzony w ramach produktu IBM MQ for Windows. Identyfikator SID może zostać użyty do uzupełnienia pola *UserIdentifier* oraz do określenia informacji autoryzacyjnych użytkownika.

Informacje o tym, w jaki sposób menedżer kolejek wypełnia pola *UserIdentifier* i *AccountingToken* , można znaleźć w opisach tych pól w polach UserIdentifier i AccountingToken.

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

Kontekst źródłowy

Kontekst źródłowy zawiera opis aplikacji, która umieszczała komunikat w kolejce, w której jest obecnie przechowywany komunikat. Deskryptor komunikatu zawiera następujące pola dla informacji o kontekście pochodzenia:

- *PutApplType* definiuje typ aplikacji, która umieszczała komunikat (na przykład transakcję CICS).
- *PutApplName* definiuje nazwę aplikacji umieszczonej w komunikacie (na przykład nazwa zadania lub transakcji).
- *PutDate* definiuje datę umieszczenia komunikatu w kolejce.
- *PutTime* definiuje czas umieszczenia komunikatu w kolejce.
- Produkt *ApplOriginData* definiuje dodatkowe informacje, które aplikacja chce uwzględnić w związku z pochodzeniem komunikatu. Na przykład można ją ustawić za pomocą odpowiednio autoryzowanych aplikacji w celu wskazania, czy dane tożsamości są zaufane.

Informacje o kontekście pochodzenia są zwykle dostarczane przez menedżer kolejek. Średni czas Greenwich (GMT) jest używany dla pól *PutDate* i *PutTime* . Zapoznaj się z opisami tych pól w polach PutDate i PutTime.

Aplikacja z wystarczającą ilością uprawnień może udostępniać własny kontekst. Umożliwia to zachowanie informacji rozliczeniowych w przypadku, gdy jeden użytkownik ma inny identyfikator użytkownika w każdym z systemów przetwarzający komunikat, z którego pochodzą.

Obiekty IBM MQ

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

Menedżery kolejek definiują właściwości (znane jako atrybuty) tych obiektów. Wartości tych atrybutów wpływają na sposób, w jaki produkt IBM MQ przetwarza te obiekty. Z poziomu aplikacji do sterowania tymi obiektami służy interfejs MQI (Message Queue Interface). Obiekty są identyfikowane przez *deskryptor obiektu* (MQOD), gdy jest on adresowany z programu.

W przypadku używania komend produktu IBM MQ do definiowania, modyfikowania lub usuwania obiektów, na przykład menedżer kolejek sprawdza, czy użytkownik posiada wymagany poziom uprawnień do wykonania tych operacji. Podobnie, jeśli aplikacja korzysta z wywołania MQOPEN w celu otwarcia obiektu, menedżer kolejek sprawdza, czy aplikacja ma wymagany poziom uprawnień, zanim zezwoli na dostęp do tego obiektu. Kontrole są przeprowadzane na podstawie nazwy otwieranego obiektu.

Pojęcia pokrewne

“Sterowanie informacjami o kontekście komunikatu” na stronie 851
W przypadku użycia wywołania MQPUT lub MQPUT1 w celu umieszczenia komunikatu w kolejce można określić, że menedżer kolejek ma dodać pewne domyślne informacje kontekstu do deskryptora komunikatu. Aplikacje, które mają odpowiedni poziom uprawnień, mogą dodawać dodatkowe informacje kontekstowe. Do sterowania informacjami kontekstowych można użyć pola opcji w strukturze MQPMO.

Odsyłacze pokrewne

“Opcje MQOPEN związane z kontekstem komunikatu” na stronie 841
Aby możliwe było powiązanie informacji kontekstowych z komunikatem podczas umieszczania go w kolejce, należy użyć jednej z opcji kontekstu komunikatu podczas otwierania kolejki.

Windows Przygotowywanie i uruchamianie aplikacji produktu Microsoft Transaction Server

Aby przygotować aplikację MTS do działania jako aplikację IBM MQ MQI client , należy postępować zgodnie z poniższa instrukcjami, aby uzyskać informacje na temat środowiska.

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

Aby przygotować aplikację MTS do uruchamiania jako aplikację 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 w sekcji “Przygotowywanie programów w języku C w programie Windows” na stronie 1123 , ale należy połączyć komponent z biblioteką mqicxa.lib zamiast z biblioteką mqic.lib.
- Jeśli komponent korzysta z klas języka C++ produktu IBM MQ , należy postępować zgodnie z instrukcjami w sekcji “Budowanie programów C++ w systemie Windows” na stronie 561 , ale należy połączyć ten komponent z biblioteką imqx23vn.lib zamiast imqc23vn.lib.
- Jeśli komponent używa powiązań języka Visual Basic dla interfejsu MQI, należy postępować zgodnie z instrukcjami w “Przygotowywanie programów Visual Basic w programie Windows” na stronie 1126 ,

ale po zdefiniowaniu projektu Visual Basic należy wpisać `MqType=3` w polu **Warunkowe argumenty kompilacji** .

- Jeśli komponent korzysta z klas automatyzacji produktu IBM MQ dla elementu ActiveX (MQAX), należy zdefiniować zmienną środowiskową GMQ_MQ_LIB o wartości `mqic32xa.dll`.

  Można zdefiniować zmienną środowiskową z poziomu aplikacji lub zdefiniować ją w taki sposób, aby jej zasięg był szeroki. Jednak zdefiniowanie go jako całego systemu może spowodować, że dowolna istniejąca aplikacja MQAX, która nie definiuje zmiennej środowiskowej z aplikacji, zachowuje się niepoprawnie.

Uwagi dotyczące projektowania aplikacji produktu IBM MQ

Po zdecydowaniu, w jaki sposób aplikacje mogą korzystać z platform i środowisk, które są dostępne dla użytkownika, 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 wniosku
Jaki jest cel Twojej aplikacji? Poniżej znajdują się odsyłacze do informacji na temat różnych typów aplikacji, które można utworzyć:

- Serwer
- Klient
- Publikowanie/subskrypcja
- Usługi WWW
- Procedury zewnętrzne, wyjścia funkcji API i usługi instalowalne

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

Język programowania
Produkt IBM MQ obsługuje wiele różnych języków programowania do 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 Twoja aplikacja będzie działać na więcej niż jednej platformie? Czy masz strategię, aby przenieść się na inną platformę z tej, której używacie dzisiaj? Jeśli odpowiedź na którekolwiek z tych pytań brzmi "tak", należy się upewnić, że kod został zakodowany w sposób niezależny od platformy.

Na przykład, jeśli używany jest kod C, kod w standardzie ANSI C. Należy użyć standardowej funkcji biblioteki 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 ma wartość paramount, gdy kod ma być używany w obu sytuacjach za pomocą programu #ifdef. Na przykład:

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

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

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

Użytkownik może skorzystać z możliwości zwiększenia dostępności, skalowalności i równoważenia obciążenia, które są możliwe w przypadku korzystania z kolejek współużytkowanych 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

Użytkownik może chcieć skorzystać z uproszczonej administracji systemu oraz zwiększyć dostępność, skalowalność i równoważenie obciążenia, które są możliwe w przypadku korzystania z klastrów.

Typy komunikatów

Można użyć datagramów dla prostych wiadomości, ale komunikaty żądania (w przypadku których oczekują odpowiedzi) na inne sytuacje. Do niektórych wiadomości można przypisać różne priorytety. Więcej informacji na temat projektowania komunikatów zawiera sekcja “Techniki projektowe dla komunikatów” na stronie 59.

Korzystanie z przesyłania komunikatów w trybie publikowania/subskrypcji lub punkt z punktem

Za pomocą przesyłania komunikatów w trybie publikowania/subskrypcji aplikacja wysyłający wysyła informacje, które chce współużytkować w komunikacie IBM MQ , do standardowego miejsca docelowego zarządzanego przez produkt IBM MQ publikowania? Subskrybuj i umożliwia IBM MQ obsługę dystrybucji tych informacji. Aplikacja docelowa nie musi wiedzieć nic o źródle otrzymywanych informacji, po prostu rejestruje zainteresowanie w jednym lub wielu tematach i otrzymuje te informacje, gdy jest ona dostępna. 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 wysyłający wysyła komunikat do określonej kolejki, skąd wie, że aplikacja odbierający ją pobierze. Aplikacja odbierający pobiera komunikaty z określonej kolejki i działa na ich zawartość. Aplikacja często będzie działać zarówno jako nadawca, jak i odbiorca, wysyłając zapytanie do innej aplikacji i otrzymując odpowiedź.

Sterowanie programami IBM MQ

Niektóre programy mogą być uruchamiane automatycznie lub gdy programy oczekują na nadejścia określonego komunikatu w kolejce (za pomocą opcji IBM MQ *wyzwalanie* patrz “Uruchamianie aplikacji produktu IBM MQ przy użyciu wyzwalaczy” na stronie 958). Alternatywnie można uruchomić inną instancję aplikacji, gdy komunikaty w kolejce nie są przetwarzane wystarczająco szybko (za pomocą opcji IBM MQ *zdarzeń instrumentacji* zgodnie z opisem w sekcji Zdarzenia instrumentacji).

Uruchamianie aplikacji na kliencie IBM MQ

Pełny interfejs MQI jest obsługiwany w środowisku klienta, a niemal każda aplikacja produktu IBM MQ napisana w języku proceduralnym może zostać zrelatowana w celu uruchomienia na serwerze IBM MQ MQI client. Powiąż aplikację na serwerze IBM MQ MQI client z biblioteką MQIC, a nie z biblioteką MQI. z/OS Pobranie (sygnał) w systemie z/OS nie jest obsługiwane.

Uwaga: Aplikacja działająca na kliencie produktu IBM MQ może jednocześnie łączyć się z więcej niż jednym menedżerem kolejek lub używać nazwy menedżera kolejek z gwiazdką (*) w wywołaniu MQCONN lub MQCONNX. Zmień aplikację, jeśli chcesz połączyć się z bibliotekami menedżera kolejek zamiast bibliotek klienta, ponieważ ta funkcja nie będzie dostępna.

Więcej informacji zawiera sekcja “Uruchamianie aplikacji w środowisku IBM MQ MQI client” na stronie 1017.

Wydajność aplikacji

Decyzje projektowe mogą mieć wpływ na wydajność aplikacji, a sugestie dotyczące zwiększenia wydajności aplikacji produktu IBM MQ można znaleźć w sekcji “Uwagi dotyczące projektowania aplikacji i wydajności” na stronie 60 IBM i i “Uwagi dotyczące projektowania i wydajności dla aplikacji produktu IBM i” na stronie 64 .

Zaawansowane techniki IBM MQ

W przypadku bardziej zaawansowanych aplikacji można użyć zaawansowanych technik IBM MQ , takich jak korelowanie odpowiedzi, a także generowanie i wysyłanie informacji o kontekście produktu IBM MQ . Więcej informacji na ten temat zawiera sekcja “Techniki projektowe dla zaawansowanych zastosowań” na stronie 62.

Zabezpieczanie danych i zachowywanie integralności

Można użyć informacji o kontekście, które są przekazywane z komunikatem do przetestowania, że komunikat został wysłany z akceptowalnego źródła. W celu zapewnienia spójności danych z innymi zasobami można użyć obiektów syncwskazujących udostępnianych przez produkt IBM MQ lub system operacyjny (szczegółowe informacje na ten temat zawiera sekcja “Zatwierdzanie i wycofywanie jednostek pracy” na stronie 946). Istnieje możliwość użycia funkcji *persistence* komunikatów produktu IBM MQ w celu zapewnienia dostarczania ważnych komunikatów.

Testowanie aplikacji produktu IBM MQ

Środowisko programowania aplikacji dla programów IBM MQ nie różni się od tego w przypadku innych aplikacji, dlatego można używać tych samych narzędzi programistycznych, jak również do narzędzi śledzenia produktu IBM MQ .

z/OS Podczas testowania aplikacji produktu CICS przy użyciu produktu IBM MQ for z/OSmożna użyć narzędzia diagnostycznego wykonania produktu CICS (CEDF). CEDF pułapuje pozycję i wyjście każdego wywołania MQI, jak również wywołania wszystkich usług produktu CICS . Ponadto w środowisku produktu CICS można napisać program obsługi wyjścia przy użyciu interfejsu API w celu udostępnienia informacji diagnostycznych przed każdym wywołaniem MQI i po jego zakończeniu. Informacje na temat sposobu wykonania tej czynności zawiera sekcja “Używanie i zapisywanie aplikacji w systemie IBM MQ for z/OS” na stronie 983.

IBM i Podczas testowania aplikacji produktu IBM i można używać standardowego debugera. Aby to uruchomić, należy użyć komendy STRDBG.

Obsługa wyjątków i błędów

Należy rozważyć sposób przetwarzania komunikatów, których nie można dostarczyć, a także sposób rozwiązywania sytuacji błędów, które są zgłaszane przez menedżera kolejek. W przypadku niektórych raportów konieczne jest ustawienie opcji raportu w tabeli MQPUT.

Pojęcia pokrewne

IBM MQ Przegląd techniczny

“Uwagi dotyczące projektowania i wydajności dla aplikacji produktu z/OS” na stronie 66
Projektowanie aplikacji jest jednym z ważniejszych czynników wpływających na wydajność. W tym temacie opisano niektóre czynniki projektowe, które są związane z wydajnością.

“Tworzenie aplikacji dla składnika IBM MQ” na stronie 5
Istnieje możliwość tworzenia aplikacji w celu wysyłania i odbierania komunikatów oraz do zarządzania menedżerami kolejek i powiązanymi zasobami. Produkt IBM MQ obsługuje aplikacje napisane w wielu różnych językach i w różnych ramach.

“Pojęcia związane z projektowaniem aplikacji” na stronie 7
Do pisania aplikacji IBM MQ można użyć wyboru języków proceduralnych lub obiektowych. Przed rozpoczęciem projektowania i pisania aplikacji produktu IBM MQ należy zapoznać się z podstawowymi pojęciami dotyczącymi produktu IBM MQ .

“Pisanie aplikacji proceduralnej w celu kolejkowania” na stronie 810
Ta sekcja zawiera informacje na temat pisania aplikacji kolejkowania, łączenia się i rozłączania z menedżerem kolejek, publikowania/subskrypcji oraz obiektów otwierających i zamykających.

“Pisanie aplikacji proceduralnych klienta” na stronie 1008
Co należy wiedzieć, aby pisać aplikacje klienckie w systemie IBM MQ , korzystając z języka proceduralnego.

“Tworzenie aplikacji produktu .NET” na stronie 566

Program IBM MQ classes for .NET umożliwia programowi napisanego w środowisku programowania .NET nawiązanie połączenia z serwerem IBM MQ jako IBM MQ MQI client lub nawiązanie bezpośredniego połączenia z serwerem IBM MQ .

“Tworzenie aplikacji C++” na stronie 537
Produkt IBM MQ udostępnia klasy języka C + + równoważne obiektom produktu IBM MQ , a niektóre dodatkowe klasy są równoważne z typami danych tablicowych. Udostępnia ona wiele funkcji, które nie są dostępne w interfejsie MQI.

“użycieIBM MQ classes for JMS” na stronie 83
IBM MQ classes for Java Message Service (IBM MQ classes for JMS) jest dostawcą JMS dostarczonym z produktem IBM MQ. Oprócz implementowania interfejsów zdefiniowanych w pakiecie javax.jms produkt IBM MQ classes for JMS udostępnia dwa zestawy rozszerzeń do interfejsu API produktu JMS .

“użycieIBM MQ classes for Java” na stronie 343
Użyj produktu IBM MQ w środowisku Java . Program IBM MQ classes for Java umożliwia aplikacji Java łączenie się z serwerem IBM MQ jako klient IBM MQ lub bezpośrednie połączenie z menedżerem kolejek produktu IBM MQ .

“Korzystanie z interfejsu modelu obiektu komponentu (klasy automatyzacji produktuIBM MQ dla elementu ActiveX)” na stronie 686
Klasy automatyzacji produktu IBM MQ dla elementów ActiveX (MQAX) to komponenty ActiveX , które udostępniają klasy, których można używać w aplikacji w celu uzyskania dostępu do produktu IBM MQ.

V 9.1.2 Określanie nazwy aplikacji w obsługiwanych językach programowania

Przed IBM MQ 9.1.2można już określić nazwę aplikacji w aplikacjach klienckich Java lub JMS .
W produkcie IBM MQ 9.1.2 ta funkcja jest rozszerzana na inne języki programowania w systemie IBM MQ for Multiplatforms.

Sposób użycia nazwy aplikacji

Nazwa aplikacji jest wyprowadzana z następujących danych:

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

Nazwa aplikacji jest również używana podczas konfigurowania śledzenia aktywności aplikacji. Domyślną nazwą aplikacji dla aplikacji innych niżJava jest obcięta nazwa pliku wykonywalnego, z wyjątkiem programów Windows.

Windows W systemie Windowsdomyślną nazwą jest pełna nazwa pliku wykonywalnego, obcięta do 28 znaków po lewej stronie.

W przypadku aplikacji Java jest to nazwa klasy poprzedzona przedrostkiem nazwy pakietu obciętą po lewej stronie od lewej do 28 znaków.

Więcej informacji na ten temat zawiera sekcja NazwaPutAppl.

W produkcie IBM MQ 9.1.2aplikacje w produkcie IBM MQ for Multiplatforms mogą ustawiać swoje nazwy aplikacji administracyjnie lub za pomocą różnych metod programowania. Umożliwia to aplikacjom udostępnianie bardziej zrozumiałej nazwy niezależnej od platformy, podczas konfigurowania śledzenia aktywności aplikacji lub w przypadku danych wyjściowych z różnych komend produktu **runmqsc** .

Produkt IBM MQ 9.1.2 dodaje możliwość równoważenia aplikacji w ramach jednolitego klastra. Do osiągnięcia tego celu używane są znaczące nazwy aplikacji.

Obsługiwane znaki

Więcej informacji na temat określania nazwy aplikacji zawiera sekcja “Zalecane znaki nazwy aplikacji” na stronie 54 .

Języki programowania

Więcej informacji na temat sposobu, w jaki aplikacje rozstrzygane w bibliotekach produktu IBM MQ w języku C oraz w innych językach programowania mogą zawierać nazwę aplikacji, zawiera sekcja “Połączenia języka programowania” na stronie 56 .

Zarządzane aplikacje produktu .NET

Informacje na temat sposobu, w jaki aplikacje zarządzane przez aplikacje .NET mogą zawierać nazwę aplikacji, zawiera sekcja “Zarządzane aplikacje .NET” na stronie 57 .

Aplikacje XMS

Informacje na temat sposobu, w jaki aplikacje XMS mogą zawierać nazwę aplikacji, zawiera sekcja “Aplikacje XMS” na stronie 58 .

Aplikacje powiązań produktu Java i produktu JMS

ULW

Informacje o tym, w jaki sposób aplikacje Java i JMS mogą zawierać nazwę aplikacji, zawiera sekcja “Aplikacje powiązań Java i JMS” na stronie 58 .

Pojęcia pokrewne

Śledzenie aktywności aplikacji

Jednolite klastry

Odsyłacze pokrewne

MQCNO

V 9.1.2 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. Szczegółowe informacje 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 V 9.1.5 i umożliwić monitorowanie zasobów aplikacji za pomocą tematów dotyczących monitorowania zasobów, nazwy aplikacji powinny zawierać tylko jednobajtowe znaki drukowalne.

Uwaga: V 9.1.5 Należy również unikać używania ukośników i znaków ampersand w nazwach aplikacji.

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
- V 9.1.5 Drukowalne znaki, które są niezmienne w kodzie EBCDIC: `+ < = > % * ' ( ) , _ - . : ; ?`

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

	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
mqclient.ini (Dotyczy tylko połączeń klienckich)	Połączenia klienckie				Połączenia klienckie		Połączenia klienckie
Java nazwa klasy		NazwaklasyJava	NazwaklasyJava				
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

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**

ULW Do MQCNO dodano dwa nowe pola, a wartość **Version** została zwiększona do 7. Więcej informacji na ten temat zawiera sekcja MQCNO.

Uwaga: z/OS Tylko aplikacje korzystające z połączeń w trybie IBM MQ classes for JMSlub IBM MQ classes for Javai trybie klienta mogą ustawiać nazwę aplikacji podczas nawiązywania połączenia z menedżerem kolejek produktu IBM MQ for z/OS .

ULW **Zmienne środowiskowe**

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

```
export MQAPPLNAME=ExampleApplName
```

Więcej informacji na ten temat zawiera sekcja Opisy zmiennych środowiskowych .

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 .NET i niezarządzanych połączeń XMS .

ULW **plik konfiguracyjny klienta**

Jeśli nie wybrano jeszcze nazwy aplikacji, a połączenie jest połączeniem klienta, w pliku konfiguracyjnym klienta (na przykład `mqclient.ini`) można określić 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 .

Przykładowy plik konfiguracyjny można znaleźć w sekcji Konfigurowanie klienta przy użyciu pliku konfiguracyjnego .

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 PutApplName .

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* , gdzie 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 można znaleźć w sekcji “Operacje zarządzane i niezarządzane w programie .NET” na stronie 661.

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 w poprzedzającym tekście, nazwa aplikacji jest automatycznie ustawiana jako nazwa pliku wykonywalnego (i w takiej części ścieżki, która zostanie dopasowana).

Aplikacje XMS

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

Właściwość fabryki połączeń

Aplikacje XMS mogą udostępniać nazwę aplikacji w fabryce połączeń za pomocą *XMSC.WMQ_APPLICATIONNAME* ("*XMSC_WMQ_APPNAME*") podobnie jak w przypadku usługi JMS. Można podać do 28 znaków.

Więcej informacji na ten temat zawierają sekcje “XMS .NET tworzenie obiektów administrowanych” na stronie 669 i “Właściwości komunikatu XMS” na stronie 677.

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.

ULW 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 nadpisania 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 przesłonięcia 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 można znaleźć w sekcji “Konfigurowanie środowiska IBM MQ dla produktu IBM MQ classes for Java” na stronie 370.

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 można znaleźć w sekcji “Konfigurowanie środowiska IBM MQ dla produktu IBM MQ classes for Java” na stronie 370.

Pojęcia pokrewne

“Konfigurowanie środowiska IBM MQ dla produktu IBM MQ classes for Java” na stronie 370
W przypadku aplikacji w celu nawiązania połączenia z menedżerem kolejek w trybie klienta aplikacja musi określić nazwę kanału, nazwę hosta i numer portu.

Odsyłacze pokrewne

MQCNO

Techniki projektowe dla komunikatów

Uwagi ułatwiające projektowanie komunikatów, w tym zagadnienia związane z selektorami i właściwościami komunikatów.

Rzeczy do rozważenia na etapie projektowania

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

Typ komunikatu, który ma zostać użyty

Czy projektujesz prostą aplikację, w której możesz wysłać wiadomość, a następnie nie podejmować dalszych działań? A może prosisz się o odpowiedź na pytanie? Jeśli zadajesz pytanie, możesz dołączyć do deskryptora komunikatu nazwę kolejki, na której chcesz otrzymać odpowiedź.

Czy chcesz, aby komunikaty żądania i odpowiedzi były synchroniczne? Oznacza to, że dla odpowiedzi zostanie ustawiony czas oczekiwania na odpowiedź na żądanie, a jeśli odpowiedź nie zostanie odezona w tym okresie, zostanie ona traktowana jako błąd.

A może wolisz pracować asynchronicznie, tak aby Twoje procesy nie musiały zależeć od występowania określonych zdarzeń, takich jak wspólne sygnały czasowe?

Inną kwestią jest to, czy masz wszystkie wiadomości wewnątrz jednostki pracy.

Przypisywanie różnych priorytetów do komunikatów

Do każdego komunikatu można przypisać wartość priorytetu, a następnie zdefiniować kolejkę w taki sposób, aby zachowała ona swoje komunikaty w kolejności ich priorytetu. W takim przypadku, gdy inny program pobierze komunikat z kolejki, zawsze otrzymuje on komunikat o najwyższym priorytecie. Jeśli kolejka nie utrzymuje swoich komunikatów w kolejności priorytetów, program pobierający komunikaty z kolejki pobierze je w kolejności, w jakiej zostały dodane do kolejki.

Programy mogą także wybierać komunikaty przy użyciu identyfikatora przypisanego do menedżera kolejek, gdy komunikat został umieszczony w kolejce. Alternatywnie można wygenerować własne identyfikatory dla każdego z komunikatów.

Wpływ zrestartowania menedżera kolejek na komunikaty

Menedżer kolejek zachowuje wszystkie komunikaty trwałe, odtwarzając je w razie potrzeby z plików dziennika produktu IBM MQ , po jego zrestartowaniu. Nietrwałe komunikaty i tymczasowe kolejki dynamiczne nie są zachowywane. Wszystkie komunikaty, które nie mają być usuwane, muszą być zdefiniowane jako trwałe po ich utworzeniu. Podczas pisania aplikacji dla systemów IBM MQ for Windows lub IBM MQ w systemach UNIX and Linux należy się upewnić, że wiadomo, w jaki sposób system został skonfigurowany w odniesieniu do przydziału plików dziennika, aby zmniejszyć ryzyko zaprojektowania aplikacji, która będzie uruchamiana w limitach plików dziennika.

z/OS Ponieważ komunikaty w kolejkach współużytkowanych (dostępne tylko w systemie IBM MQ for z/OS) są przechowywane w narzędziu CF (Coupling Facility), komunikaty nietrwałe są

zachowywane po restarcie menedżera kolejek tak długo, jak długo system CF pozostaje dostępny. Jeśli działanie systemu CF nie powiedzie się, komunikaty nietrwałe zostaną utracone.

Udzielanie informacji o sobie na rzecz odbiorcy wiadomości
Zwykle menedżer kolejek ustawia ID użytkownika, ale odpowiednio autoryzowane aplikacje mogą również ustawić to pole, dzięki czemu można uwzględnić własny identyfikator użytkownika i inne informacje, które program odbierający może wykorzystać do celów księgowych lub zabezpieczeń.

Ilość kolejek odbiorczy

Jeśli może być konieczne umieszczenie komunikatu w kilku kolejkach, można opublikować go w temacie lub na liście dystrybucyjnej.

Jeśli może być konieczne umieszczenie komunikatu w kilku kolejkach, można opublikować go w temacie.

Selektory i właściwości komunikatu

Komunikaty mogą zawierać metadane powiązane z nimi wraz z ładunkiem komunikatu głównego. Te właściwości komunikatu mogą być przydatne w dostarczaniu dodatkowych danych.

Istnieją dwa aspekty dotyczące tych dodatkowych danych, które należy wiedzieć o:

- Właściwości nie podlegają ochronie Advanced Message Security (AMS). Aby użyć AMS w celu ochrony danych, należy umieścić go w ładunku, a nie we właściwościach komunikatu.
- Właściwości mogą być używane do wykonywania wyboru komunikatów.

Ważne jest, aby pamiętać, że korzystanie z selektorów powoduje podział standardowej konwencji komunikatów w pierwszej kolejności na początku. Ponieważ menedżer kolejek jest zoptymalizowany pod kątem tego obciążenia, dostarczanie 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 jest kolejka, tym bardziej złożony jest selektor i mniejsze prawdopodobieństwo, że selektor zgodny z komunikatem będzie miał negatywny wpływ na wydajność.

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

Uwaga: Produkt IBM MQ classes for Java nie obsługuje użycia selektorów, jeśli użytkownik chce korzystać z selektorów, które należy wykonać za pośrednictwem interfejsu API produktu JMS .

Uwagi dotyczące projektowania aplikacji i wydajności

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

Oto kilka pomysłów, które pomogą Ci zaprojektować wydajne aplikacje:

- Zaprojektuj aplikację w taki sposób, aby przetwarzanie było wykonywane równolegle z czasem myślenia użytkownika:
  - Wyświetl panel i pozwól użytkownikowi na rozpoczęcie wpisywania w czasie, gdy aplikacja nadal jest inicjowana.
  - Uzyskaj dane, które są potrzebne równolegle z różnych serwerów.
- Należy utrzymywać otwarte połączenia i kolejki, jeśli ich ponowne użycie zamiast wielokrotnego otwierania i zamykania, łączenia i rozłączania jest ponownie otwarte.
- Jednak aplikacja serwera, która jest umieszczana tylko w jednym komunikacie, powinna używać parametru MQPUT1.
- Menedżery kolejek są zoptymalizowane pod względem wielkości komunikatów o wielkości od 4 kB do 100 kB. Bardzo duże komunikaty są nieefektywne; prawdopodobnie lepiej jest wysłać 100

komunikatów o wielkości 1 MB każdy, niż jeden komunikat o wielkości 100 MB. Bardzo małe komunikaty są również nieefektywne. Menedżer kolejek wykonuje tę samą ilość pracy dla komunikatu jednobajtowego, co w przypadku komunikatu o wielkości 4 kB.

- Komunikaty należy przechowywać w jednostce pracy, tak aby mogły być zatwierdzane lub wycofane jednocześnie.
- Użyj opcji nietrwałej dla komunikatów, które nie muszą być odtwarzalne.
- Jeśli konieczne jest wysłanie komunikatu do kilku 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 z aplikacji, należy wysłać tylko istotne dane w komunikacie. Na przykład, w żądaniu obciążenia rachunku bankowego, jedyną informacją, która może być przekazywana z klienta do aplikacji serwera jest numer konta i kwota obciążenia.

Wpływ trwałości komunikatów

Komunikaty trwałe są zwykle rejestrowane. Protokołowanie komunikatów zmniejsza wydajność aplikacji, dlatego należy używać trwałych komunikatów tylko dla istotnych danych. Jeśli dane w komunikacie mogą zostać usunięte, jeśli menedżer kolejek zostanie zatrzymany lub nie powiedzie się, należy użyć nietrwałego komunikatu.

z/OS Operacje MQPUT i MQGET dla komunikatów trwałych będą blokują się, gdy nie ma wystarczającej ilości miejsca w dzienniku odtwarzania do zarejestrowania operacji. Taki warunek jest wskazyany w protokole zadania menedżera kolejek za pomocą komunikatów CSQJ110E i CSQJ111A. Upewnij się, że istnieją procesy monitorowania, dzięki czemu warunki te są zarządzane i unikane.

Wyszukiwanie konkretnego komunikatu

Wywołanie MQGET zwykle pobiera pierwszy komunikat z kolejki. Jeśli w deskryptorze komunikatu używane są komunikaty i identyfikatory korelacji (*MsgId* i *CorrelId*) w celu określenia konkretnego komunikatu, menedżer kolejek musi przeszukać kolejkę do momentu znalezienia 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, powiększaj i zmniejszaj bufory dynamicznie, tak aby odpowiadała typowi wielkości komunikatu. Jeśli aplikacja zgłosi wywołanie MQGET, które nie powiedzie się, ponieważ bufor jest za mały, zwracana jest wielkość danych komunikatu. Dodaj kod do aplikacji w taki sposób, aby bufor został odpowiednio zmieniony, a wywołanie MQGET zostało ponownie wysłane.

Uwaga: Jeśli atrybut **MaxMsgLength** nie zostanie ustawiony jawnie, wartością domyślną będzie 4 MB, co może być bardzo nieefektywne, jeśli jest to używane do wywierania wpływu na wielkość buforu aplikacji.

Częstotliwość punktów synchronizacji

Programy, które emitują bardzo dużą liczbę wywołań MQPUT lub MQGET w punkcie synchronizacji, bez ich zatwierdzania, mogą powodować problemy z wydajnością. Przydzielone kolejki mogą zapełniać komunikaty, które są obecnie niedostępne, podczas gdy inne zadania mogą oczekiwać na pobranie tych komunikatów. Ma to wpływ na pamięć masową, a także na wątki powiązane z zadaniami, które próbują uzyskać komunikaty.

Użycie wywołania MQPUT1

Wywołania MQPUT1 należy używać tylko wtedy, gdy istnieje pojedynczy komunikat, który ma zostać umieszczony w kolejce. Jeśli chcesz umieścić więcej niż jeden komunikat, użyj wywołania MQOPEN, po którym następuje seria wywołań MQPUT i pojedyncze wywołanie MQCLOSE.

Liczba używanych wątków

Windows W przypadku produktu IBM MQ for Windowsaplikacja może wymagać dużej liczby wątków. Dla każdego procesu menedżera kolejek przydzielana jest maksymalna dozwolona liczba wątków aplikacji.

Aplikacje mogą używać zbyt wielu wątków. Zastanów się, czy aplikacja bierze pod uwagę tę możliwość i że podejmuje działania, aby zatrzymać lub zgłosić ten rodzaj wystąpienia.

Umieszczanie trwałych komunikatów w punkcie synchronizacji

Komunikaty trwałe powinny być umieszczane w punkcie synchronizacji i muszą być umieszczane w punkcie synchronizacji. Dzieje się tak dlatego, że podczas pobierania trwałego komunikatu poza punktem synchronizacji, jeśli operacja pobierania nie powiedzie się, nie ma możliwości, aby aplikacja wiedziała, czy komunikat został wysłany z kolejki, czy też nie. Jeśli komunikat został wyświetlony, oznacza to, że został on również utracony. Podczas pobierania trwałych komunikatów w punkcie synchronizacji, jeśli coś się nie powiedzie, transakcja zostanie wycofana, a komunikat trwały nie zostanie utracony, ponieważ nadal znajduje się w kolejce.

Podobnie, umieszczając komunikaty trwałe, należy umieścić je w punkcie synchronizacji. Innym powodem umieszczania i pobierania trwałych komunikatów w punkcie synchronizacji jest to, że trwały kod komunikatu w produkcie IBM MQ jest w dużej mierze zoptymalizowany pod kątem punktu synchronizacji. Dlatego umieszczenie i uzyskiwanie trwałych komunikatów w punkcie synchronizacji jest szybsze niż wprowadzanie i pobieranie trwałych komunikatów poza punktem synchronizacji.

Jeśli aplikacja wykonuje umieszczanie trwałych komunikatów poza punktem synchronizacji, menedżer kolejek sprawdza, czy może utworzyć niejawny punkt synchronizacji w imieniu aplikacji. Jeśli menedżer kolejek może to zrobić, obejmuje on umieszczanie wewnątrz tego punktu synchronizacji, a następnie zatwierdza go automatycznie. Więcej informacji na ten temat zawiera sekcja “Niejawny punkt synchronizacji na wielu platformach” na stronie 955 .

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

Jeśli aplikacja otrzymuje komunikaty i nie ma informacji o tym, czy są one trwałe, czy nie, można użyć opcji GMO MQGMO_SYNCPOINT_IF_PERSISTENT.

Techniki projektowe dla zaawansowanych zastosowań

Projektując bardziej zaawansowane aplikacje, istnieje kilka technik, które mogą być takie, jak oczekiwanie na komunikaty, korelowanie odpowiedzi, ustawianie i korzystanie z informacji kontekstowych, uruchamianie aplikacji automatycznie, generowanie raportów i usuwanie powinowactw komunikatów podczas korzystania z technologii klastrowej.

W przypadku prostej aplikacji IBM MQ należy zdecydować, które obiekty produktu IBM MQ mają być używane w aplikacji, a także typy komunikatów, które mają być używane. W przypadku bardziej zaawansowanych aplikacji warto skorzystać z niektórych technik wprowadzonych w poniższych sekcjach.

Oczekiwanie na komunikaty

Program obsługujący kolejkę może czekać na komunikaty przez:

- Trwa oczekiwanie na nadejście komunikatu lub upłynie określony przedział czasu (patrz “Oczekiwanie na komunikaty” na stronie 889).
- z/OS Tylko w systemie IBM MQ for z/OS : ustawianie sygnału w taki sposób, aby program był powiadamiany po nadejściu komunikatu. Więcej informacji na ten temat zawiera sekcja “sygnalizowanie” na stronie 890.
- Ustanawianie wyjścia wywołania zwrotnego, które ma być sterowane po nadejściu komunikatu; patrz “Asynchroniczne wykorzystanie komunikatów produktu IBM MQ” na stronie 42.
- Wykonywanie okresowych wywołań w kolejce w celu sprawdzenia, czy komunikat został wysłany (*odpytywanie*). Zwykle nie jest to zalecane, ponieważ może mieć wpływ na wydajność.

Korelowanie odpowiedzi

W aplikacjach IBM MQ , gdy program odbierze komunikat, który żąda jego wykonania, program zwykle wysyła do requestera co najmniej jeden komunikat odpowiedzi.

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

Ustawianie i używanie informacji kontekstowych

Informacje o kontekście są używane do tworzenia powiązań komunikatów z użytkownikiem, który je wygenerował, a także do identyfikowania aplikacji, która wygenerował komunikat. Takie informacje są przydatne w celu zapewnienia bezpieczeństwa, rozliczania, kontroli i określania problemów.

Podczas tworzenia komunikatu można określić opcję, która żąda, aby menedżer kolejek powiąże domyślne informacje kontekstu z komunikatem.

Więcej informacji na temat używania i ustawiania informacji o kontekście zawiera sekcja “Kontekst komunikatu” na stronie 47.

Automatyczne uruchamianie programów IBM MQ

Użyj opcji IBM MQ *wyzwalanie* , aby uruchomić program automatycznie, gdy komunikaty pojawiają się w kolejce.

Można ustawić warunki wyzwalacza w kolejce, tak aby program uruchamiał tę kolejkę:

- Za każdym razem, gdy komunikat pojawia się w kolejce
- Gdy pierwszy komunikat pojawia się w kolejce
- Gdy liczba komunikatów w kolejce osiągnie predefiniowaną liczbę

Więcej informacji na temat wyzwalania zawiera sekcja “Uruchamianie aplikacji produktu IBM MQ przy użyciu wyzwalaczy” na stronie 958. Wyzwalanie to tylko jeden ze sposobów automatycznego uruchamiania programu. Na przykład można uruchomić program automatycznie na czasomierz za pomocą narzędzi innych niżIBM MQ .

Multi W systemie Wiele platformprodukt IBM MQ może definiować obiekty usług w celu uruchamiania programów IBM MQ podczas uruchamiania menedżera kolejek. Patrz sekcja Obiekty usług.

Generowanie raportów IBM MQ

W ramach aplikacji można zażądać następujących raportów:

- Raporty dotyczące wyjątków
- Sprawozdania z wygaśnięcia
- Raporty potwierdzenia w momencie przybycia (COA)
- Raporty potwierdzenia dostarczenia (COD)

- Raporty z powiadomieniem o działaniu pozytywnym (PAN)
- Raporty powiadomień o działaniu negatywnym (NAN)

Są one opisane w sekcji "Komunikaty raportów" na stronie 19.

Klastry i powinowactwa komunikatów

Przed rozpoczęciem korzystania z klastrów z wieloma definicjami dla tej samej kolejki należy sprawdzić aplikacje, aby sprawdzić, czy istnieją jakieś wymagające wymiany powiązanych komunikatów.

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

Na przykład mogą być używane dwa aplikacje, które polegają na serii komunikatów przepływających między nimi w formie pytań i odpowiedzi. Ważne może być, aby wszystkie pytania były wysyłane do tego samego menedżera kolejek oraz 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 tak się składa, że tylko w tym przypadku jest hostem instancji odpowiedniej kolejki.

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

Więcej informacji na ten temat zawiera sekcja Obsługa powinowactw komunikatów.

Uwagi dotyczące projektowania i wydajności dla aplikacji produktu IBM i

Te informacje umożliwiają zrozumienie sposobu, w jaki projektowanie aplikacji, wątki i pamięć masowa mogą mieć wpływ na wydajność.

Informacje te są podzielone na dwie sekcje:

- "Uwagi dotyczące projektowania aplikacji" na stronie 64
- "Szczególne problemy z wydajnością" na stronie 65

Uwagi dotyczące projektowania aplikacji

Istnieje wiele sposobów, w jaki kiepski projekt programu może mieć wpływ na wydajność. Problemy te mogą być trudne do wykrycia, ponieważ może się wydawać, że program może działać poprawnie, a jednocześnie wpływa na wydajność innych zadań. W poniższych sekcjach opisano kilka problemów związanych z programami wywołując IBM MQ for IBM i .

Więcej informacji na temat projektowania aplikacji zawiera sekcja "Uwagi dotyczące projektowania aplikacji produktu IBM MQ" na stronie 50.

Wpływ długości komunikatu
Chociaż program IBM MQ for IBM i umożliwia wstrzymanie komunikatów do 100 MB danych, ilość danych w komunikacie wpływa na wydajność aplikacji, która przetwarza komunikat. Aby osiągnąć najlepsze wyniki z aplikacji, należy wysłać tylko istotne dane w komunikacie; na przykład w żądaniu obciążenia rachunku bankowego, jedynymi informacjami, które mogą być przekazywane od klienta do aplikacji serwera jest numer konta i kwota obciążenia.

Wpływ trwałości komunikatów
Komunikaty trwałe są kronikowane. Kronikowanie komunikatów zmniejsza wydajność aplikacji, dlatego należy używać trwałych komunikatów tylko dla istotnych danych. Jeśli dane w komunikacie mogą zostać usunięte, jeśli menedżer kolejek zostanie zatrzymany lub nie powiedzie się, należy użyć nietrwałego komunikatu.

Wyszukiwanie konkretnego komunikatu
Wywołanie MQGET zwykle pobiera pierwszy komunikat z kolejki. Jeśli w deskryptorze komunikatu używane są komunikaty i identyfikatory korelacji (*MsgId* i *CorrelId*) w celu określenia konkretnego

komunikatu, 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, aby określić wielkość komunikatu, aplikacja może użyć wywołania MQGET z polem *BufferLength* ustawionym na zero, dzięki czemu, mimo że wywołanie nie powiedzie się, zwraca on wielkość danych komunikatu. Aplikacja może następnie powtórzyć wywołanie, określając identyfikator komunikatu mierzony w jego pierwszym wywołaniu oraz bufor o prawidłowej wielkości. Jeśli jednak istnieją inne aplikacje obsługujące tę samą kolejkę, może się okazać, że wydajność aplikacji jest zmniejszona, ponieważ jej drugie wywołanie MQGET jest czasochłonne podczas wyszukiwania komunikatu, który został pobrany przez inną aplikację 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 mogą zostać zaakceptowane przez kolejkę, a następnie użyj tej wartości w wywołaniu MQGET. Maksymalna wielkość komunikatów dla kolejki jest przechowywana w atrybucie **MaxMsgLen** w kolejce. Ta metoda może jednak używać dużych ilości pamięci masowej, ponieważ wartość tego atrybutu kolejki może być wartością maksymalną dopuszczalną przez produkt IBM MQ for IBM i, która może być większa niż 2 GB.

Częstotliwość punktów synchronizacji
Programy, które emitują wiele wywołań MQPUT w punkcie synchronizacji, bez ich zatwierdzania, mogą powodować problemy z wydajnością. Przydzielone kolejki mogą zapełniać komunikaty, które są obecnie bezużyteczne, podczas gdy inne zadania mogą oczekiwać na pobranie tych komunikatów. Ten problem ma wpływ na pamięć masową, a w kontekście wątków powiązanych z zadaniami, które próbują uzyskać komunikaty.

Użycie wywołania MQPUT1
Wywołania MQPUT1 należy używać tylko wtedy, gdy istnieje pojedynczy komunikat, który ma zostać umieszczony w kolejce. Jeśli chcesz umieścić więcej niż jeden komunikat, użyj wywołania MQOPEN, po którym następuje seria wywołań MQPUT i pojedyncze wywołanie MQCLOSE.

Liczba używanych wątków
Aplikacja może wymagać wielu wątków. Dla każdego procesu menedżera kolejek przydzielana jest maksymalna dozwolona liczba wątków. Jeśli niektóre aplikacje są kłopotliwe, może to być spowodowane ich projektowaniem za pomocą zbyt dużej liczby wątków. Zastanów się, czy aplikacja bierze pod uwagę tę możliwość i że podejmuje działania, aby zatrzymać lub zgłosić ten rodzaj wystąpienia. Maksymalna liczba wątków, które zezwala na IBM i , wynosi 4,095. Wartością domyślną jest 64. Program IBM MQ udostępnia maksymalnie 63 wątki do jego procesów.

Szczególne problemy z wydajnością

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

Problemy z pamięcią
Jeśli zostanie wyświetlony komunikat systemowy `CPF0907. Serious storage condition may exist` , to możliwe jest wypełnienie obszaru powiązanego z menedżerami kolejek produktu IBM MQ for IBM i .

Czy aplikacja lub produkt 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. Ten powolny proces może być również spowodowany przez problem z wydajnością. Być może dlatego, że Twój system działa w pobliżu granic swojej zdolności. Ten typ problemu jest prawdopodobnie najgorszy w szczytowym czasie ładowania systemu, zazwyczaj w połowie dnia rano i w połowie popołudnia. (Jeśli sieć rozciąga się w więcej niż jednej strefie czasowej, szczytowe obciążenie systemu może się wydawać, że nastąpi to w innym czasie).

Jeśli nie jest to zależne od ładowania systemu, ale zdarza się to czasami, gdy system jest ładnie ładowany, to źle zaprojektowany program użytkowy jest prawdopodobnie obwiniony. Ten problem może być objawiony jako problem, który występuje tylko wtedy, gdy dostęp do niektórych kolejek jest uzyskiwany.

QTOTJOB i QADLTOTJ to wartości systemowe, które warto zbadać.

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

- Jeśli system jest wolny, aby odpowiedzieć na komendy MQSC.
- Jeśli powtórzone wyświetlanie głębokości kolejki wskazuje, że kolejka jest przetwarzana powoli, dla aplikacji, z którą należy oczekiwać dużej aktywności kolejki.
- Czy śledzenie IBM MQ jest uruchomione?

Linux Uwagi dotyczące projektowania aplikacji produktu Linux on POWER Systems - Little Endian

Ponieważ produkt Linux on POWER Systems - Little Endian obsługuje tylko aplikacje 64-bitowe, w produkcie IBM MQ nie jest dostępna obsługa dla aplikacji 32-bitowych.

Pojęcia pokrewne

“Uwagi dotyczące projektowania aplikacji produktu IBM MQ” na stronie 50

Po zdecydowaniu, w jaki sposób aplikacje mogą korzystać z platform i środowisk, które są dostępne dla użytkownika, należy zdecydować, w jaki sposób korzystać z funkcji oferowanych przez produkt IBM MQ.

z/OS Uwagi dotyczące projektowania i wydajności dla aplikacji produktu z/OS

Projektowanie aplikacji jest jednym z ważniejszych czynników wpływających na wydajność. W tym temacie opisano niektóre czynniki projektowe, które są związane z wydajnością.

Istnieje wiele sposobów, w jaki kiepski projekt programu może mieć wpływ na wydajność. Problemy te mogą być trudne do wykrycia, ponieważ może się wydawać, że program może działać poprawnie, a jednocześnie wpływa na wydajność innych zadań. W poniższych sekcjach przedstawiono kilka problemów specyficznych dla programów wykonających wywołania MQI.

Więcej informacji na temat projektowania aplikacji zawiera sekcja “Uwagi dotyczące projektowania aplikacji produktu IBM MQ” na stronie 50.

Wpływ długości komunikatu

Chociaż program IBM MQ for z/OS umożliwia wstrzymanie komunikatów do 100 MB danych, ilość danych w komunikacie wpływa na wydajność aplikacji, która przetwarza komunikat. Aby uzyskać najlepszą wydajność z aplikacji, należy wysłać tylko istotne dane w komunikacie. Na przykład, w żądaniu obciążenia rachunku bankowego, jedyną informacją, która może być przekazana z klienta do aplikacji serwera jest numer konta i kwota do obciążenia.

Wpływ trwałości komunikatów

Komunikaty trwałe są rejestrowane. Protokołowanie komunikatów zmniejsza wydajność aplikacji, dlatego należy używać trwałych komunikatów tylko dla istotnych danych. Jeśli dane w komunikacie mogą zostać usunięte, jeśli menedżer kolejek zostanie zatrzymany lub nie powiedzie się, należy użyć nietrwałego komunikatu.

Dane dla trwałych komunikatów są zapisywane w buforach dziennika. Bufory te są zapisywane w zestawach danych dziennika, gdy:

- Występuje zatwierdzanie
- Komunikat został pobrany z punktu synchronizacji lub został on wyjęty z punktu synchronizacji
- Bufory WRTHRSH są wypełnione

Przetwarzanie wielu komunikatów w jednej jednostce pracy może spowodować mniejsze wejście/wyjście, niż w przypadku, gdy komunikaty były przetwarzane po jednym dla każdej jednostki pracy, lub poza punktem synchronizacji.

Wyszukiwanie konkretnego komunikatu

Wywołanie MQGET zazwyczaj pobiera pierwszy komunikat z kolejki. Jeśli używane są identyfikatory komunikatów i korelacji (**MsgId** i **CorrelId**), w deskryptorze komunikatu, aby określić konkretny komunikat, 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 program IBM MQ może mieć możliwość skanowania całej kolejki.

Za pomocą atrybutu kolejki **IndexType** można określić, że menedżer kolejek ma utrzymywać indeks, który może być używany do zwiększania szybkości operacji MQGET w kolejce. Jednak istnieje niewielkie zmniejszenie wydajności dla zachowania indeksu, dlatego należy go wygenerować tylko wtedy, gdy jest konieczne jego użycie. Można utworzyć indeks identyfikatorów komunikatów lub identyfikatorów korelacji, albo nie można zbudować indeksu dla kolejek, w których komunikaty są pobierane sekwencyjnie. Spróbuj użyć wielu różnych wartości kluczy, a nie partii o tej samej wartości. Na przykład: Balance1, Balance2i Balance3, a nie trzy z równowag. W przypadku kolejek współużytkowanych należy mieć poprawną wartość **IndexType**. Szczegółowe informacje na temat atrybutu kolejki **IndexType** zawiera sekcja IndexType.

Aby uniknąć wpływu na czas restartu menedżera kolejek przy użyciu poindeksowanych kolejek, 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** oraz inne atrybuty obiektów znajdują się w sekcji Atrybuty obiektów.

Kolejki zawierające komunikaty o różnych długościach

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

Jeśli zostanie wydana komenda MQGET z indeksem o długości zero, zostanie zwrócona wielkość komunikatu, a aplikacja będzie mogła pobrać bufor o tej wielkości i ponownie wydać komendę get. Jeśli istnieje wiele aplikacji przetwarzaających kolejkę, inna aplikacja mogła już przetworzyć ten komunikat, gdy pierwotny wniosek został ponownie wydany. Jeśli sporadycznie istnieją duże komunikaty, może być konieczne uzyskanie dużego buforu tylko dla tych komunikatów, a następnie zwolnienie go po przetworzeniu komunikatu. Powinno to pomóc zmniejszyć problemy związane 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 mogą zostać zaakceptowane przez kolejkę, a następnie użyj tej wartości w wywołaniu MQGET . Maksymalna wielkość komunikatów dla kolejki jest przechowywana w atrybucie **MaxMsgL** w kolejce. Ta metoda może jednak używać dużych ilości pamięci masowej, ponieważ wartość **MaxMsgL** może być tak wysoka, jak 100 MB, a maksymalna dozwolona w programie IBM MQ for z/OS.

Uwaga: Parametr **MaxMsgL** można zmniejszyć po umieszczanie dużych komunikatów w kolejce. Na przykład można umieścić komunikat 100 MB, a następnie ustawić wartość **MaxMsgL** na 50 bajtów. Oznacza to, że nadal możliwe jest uzyskanie większych wiadomości niż oczekiwała aplikacja.

Częstotliwość punktów synchronizacji

Programy, które wywołują wiele wywołań MQPUT w punkcie synchronizacji, bez ich zatwierdzania, mogą powodować problemy z wydajnością. Przydzielone kolejki mogą zapełniać komunikaty, które są obecnie

bezużyteczne, podczas gdy inne zadania mogą oczekiwać na pobranie tych komunikatów. Ma to wpływ na pamięć masową, a także na wątki związane z zadaniami, które próbują uzyskać komunikaty.

Co do zasady, jeśli wiele aplikacji przetwarza kolejkę, zwykle otrzymujesz najlepszą wydajność, gdy masz albo

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

dla każdego punktu synchronizacji. Jeśli istnieje tylko jedna aplikacja przetwarzający kolejkę, konieczne jest posiadanie większej liczby komunikatów dla każdej jednostki pracy.

Istnieje możliwość ograniczenia liczby komunikatów, które zadanie może pobrać lub umieścić w pojedynczej jednostce odtwarzania z atrybutem menedżera kolejek produktu **MAXUMSGS** . Więcej informacji na temat tego atrybutu zawiera opis komendy **ALTER QMGR** w sekcji Komendy MQSC.

Korzyści z wywołania MQPUT1

Wywołania MQPUT1 należy używać tylko wtedy, gdy istnieje pojedynczy komunikat do umieszczenia w kolejce. Jeśli ma zostać umieszczony więcej niż jeden komunikat, należy użyć wywołania MQOPEN , po którym następuje seria wywołań MQPUT i pojedynczego wywołania MQCLOSE .

Liczba komunikatów, które mogą zawierać menedżer kolejek

Kolejki lokalne

Liczba komunikatów lokalnych, które mogą być przechowywane przez menedżera kolejek, to w zasadzie wielkość zestawów stron. Użytkownik może mieć do 100 zestawów stron (mimo że jest to zalecany zestaw stron 0, a zestaw stron 1 dotyczy obiektów i kolejek związanych z systemem). Istnieje możliwość użycia zestawu stron z rozszerzonym formatem i zwiększenia możliwości zestawu stron.

Kolejki współużytkowane

Wielkość kolejek współużytkowanych zależy od wielkości narzędzia CF. Produkt IBM MQ używa struktur listy CF, w których podstawowe jednostki pamięci są pozycjami i elementami. Każdy komunikat jest składowany jako 1 pozycja i wiele elementów zawierających powiązane dane komunikatu MQMD i inne dane komunikatu. Liczba elementów skonsumowanych przez pojedynczy komunikat zależy od wielkości komunikatu, a dla CFLEVEL (5)-reguły odciążania w czasie MQPUT. Jeśli dane komunikatu są przenoszone do partycji Db2 lub SMDS, wymagane jest mniej elementów. Dostęp do danych komunikatu jest wolniejszy, gdy komunikat został odciążony. Więcej informacji na temat wydajności i narzutu procesora związanego z przesuniętym komunikatem można znaleźć w sekcji Performance Supportpac MP1H .

Co wpływa na wydajność

Wydajność może oznaczać, jak szybkie komunikaty mogą być przetwarzane, a także może oznaczać, ile czasu procesora jest potrzebny na komunikat.

Co ma wpływ na to, jak szybkie komunikaty mogą być przetwarzane

W przypadku komunikatów trwałych największym uderzeniem jest szybkość zestawów danych dziennika. Szybkość zestawów danych dziennika zależy od urządzenia DASD, w którym są one dostępne. W związku z tym należy zwrócić uwagę na umieszczenie zestawu danych dziennika na niskim zużyciu woluminów w celu zmniejszenia rywalizacji. Rozsianie dzienników programu MQ poprawia wydajność dziennika, gdy na we/wy znajduje się wiele stron zapisanych na potrzeby operacji we/wy. Z kolei High Performance Fibre connection (zHPF) ma również znaczny wpływ na czas odpowiedzi 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. W celu planowania należy rozważyć, czy kolejka jest zablokowana dla całego żądania. Jeśli więc czas na odkładanie to 100 mikrosekund, a masz więcej niż 10 000 żądań, to możesz doświadczyć opóźnień. Można osiągnąć lepsze wyniki niż to w praktyce, ale jest to dobra ogólna zasada. W celu zwiększenia wydajności można użyć różnych kolejek.

Możliwe przyczyny tego działania mogą być następujące:

- Użyj wspólnej kolejki odpowiedzi, której używa każda transakcja CICS
- każda transakcja CICS otrzymuje unikalną odpowiedź na kolejkę
- Odpowiedź na kolejkę dla regionu CICS i wszystkie transakcje w regionie CICS używają tej kolejki.

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

Jeśli komunikaty muszą zostać odczytane z zestawu stron, będą one wolniejsze w porównaniu z tym, kiedy komunikaty znajdują się w puli buforów. Jeśli masz więcej wiadomości niż zmieścisz się do puli buforów, to będą one rozlewać się na dysk. W związku z tym należy upewnić się, że pula buforów jest wystarczająco duża, aby można było znaleźć krótkie komunikaty. Jeśli komunikaty są przetwarzane wiele godzin później, prawdopodobnie będą one rozlewać na dysk, dlatego należy oczekiwać, że komunikaty będą wolniejsze niż w przypadku, gdy znajdowały się w puli buforów.

W przypadku kolejki współużytkowanej szybkość komunikatów zależy od szybkości narzędzia CF. System CF w procesorze fizycznym może być szybszy od zewnętrznego systemu CF. Czas odpowiedzi systemu CF zależy od tego, jak zajęte jest system CF. Na przykład w systemach Hursley, kiedy system CF był o 17% zajęty, czas odpowiedzi wynosił 14 mikrosekund. Kiedy system CF był zajęty w 95%, czas odpowiedzi wynosił 45 mikrosekund.

Jeśli żądania MQ używają dużo czasu procesora, może to mieć wpływ na sposób przetwarzania szybkich komunikatów. Ponieważ, jeśli partycja logiczna (LPAR) jest ograniczona przez procesor, aplikacje będą opóźniane w oczekiwaniu na procesor.

Ilość procesora na komunikat

W ogólności większe komunikaty używają więcej procesora, więc należy unikać dużych (x MB) komunikatów, jeśli to możliwe.

Podczas pobierania konkretnych komunikatów z kolejek, kolejka powinna być poindeksowana, aby menedżer kolejek mógł przejść bezpośrednio do komunikatu (i w ten sposób uniknąć potencjalnie całego skanowania kolejki). Jeśli kolejka nie jest poindeksowana, to kolejka jest skanowana od początku szukając komunikatu. Jeśli w kolejce znajduje się 1000 komunikatów, może być konieczne skanowanie wszystkich 1000 komunikatów. Wynikiem jest wiele niepotrzebnego użycia procesora.

Kanały używające protokołu TLS mają dodatkowy koszt ze względu na szyfrowanie wiadomości.

W programie MQ V7 oprócz **CORRELID** lub **MSGID**można wybierać komunikaty przy użyciu łańcucha selektora. Każdy komunikat musi być przeglądany, więc jeśli w kolejce jest wiele komunikatów, to jest to kosztowne.

Aplikacja do wykonania komendy OPEN PUT CLOSE jest bardziej wydajna, niż PUT1 PUT1.

Wyzwalanie w programie CICS

Gdy szybkość wysyłania komunikatów dla wyzwalanej kolejki jest niska, jest on wydajny, aby najpierw użyć wyzwalacza. Gdy szybkość wysyłania komunikatów jest większa niż 10 komunikatów na sekundę, bardziej wydajne jest wyzwalanie pierwszej transakcji, a następnie proces transakcji jest komunikatem i wyświetlany jest następny komunikat itd. Jeśli komunikat nie dotarł do krótkiego okresu (powiedzmy między 0.1 i 1 sekunda), transakcja kończy się. W przypadku wysokiej przepustowości może być konieczne uruchomienie wielu transakcji w celu przetworzenia komunikatów i uniemożliw tworzenie komunikatów. Dla każdego wytworzonego komunikatu wyzwalacza wymaga to umieszczenia komunikatu wyzwalacza i uzyskania komunikatu wyzwalacza, który w efekcie podwaja koszt komunikatu.

Liczba obsługiwanych połączeń lub jednocześnie pracujących użytkowników

Każde połączenie korzysta z wirtualnej pamięci masowej w menedżerze kolejek, co powoduje, że im więcej użytkowników będzie więcej niż w pamięci masowej, tym więcej będzie używana. Jeśli potrzebna jest bardzo duża pula buforów i duża liczba użytkowników, można ograniczyć wielkość pamięci wirtualnej, co może wymagać zmniejszenia wielkości pul buforów.

Jeśli zabezpieczenia są używane, menedżer kolejek buforuje informacje w menedżerze kolejek przez długi czas. Wpływa to na ilość wirtualnej pamięci masowej, która jest używana w menedżerze kolejek.

CHINIT może obsługiwać do około 10 000 połączeń. Jest to ograniczone przez wirtualną pamięć masową. Jeśli połączenie korzysta z większej ilości pamięci masowej, na przykład przy użyciu protokołu TLS, pamięć masowa na połączenie zwiększa się, co oznacza, że **CHINIT** może obsługiwać mniej połączeń. W przypadku przetwarzania dużych komunikatów będzie ono wymagał więcej pamięci dla buforów w partycji **CHINIT**, dzięki czemu produkt **CHINIT** może obsługiwać mniej komunikatów.

Połączenia ze zdalnym menedżerem kolejek są bardziej wydajne niż połączenia klienckie. Na przykład każde żądanie klienta MQ wymaga dwóch przepływów sieciowych (jeden dla żądania i jeden dla odpowiedzi). W przypadku kanału do zdalnego menedżera kolejek może być 50 wysyłanych przez sieć, zanim odpowiedź zostanie wycofana. Jeśli rozważana jest duża sieć kliencka, bardziej wydajne może być użycie menedżera kolejek koncentratora w polu rozproszonym i jeden kanał przychodzący do koncentratora i z niego wyjmowany.

Inne czynniki wpływające na wydajność

Zestaw danych dziennika powinien mieć wielkość co najmniej 1000 cylindrów. Jeśli dzienniki są mniejsze niż ten, działanie punktu kontrolnego może być zbyt częste. W systemie pracowitym punkt kontrolny zwykle powinien być co 15 minut lub dłużej, na bardzo wysokim poziomie, który może być mniejszy niż ten. Gdy punkt kontrolny wystąpi, pule buforów są skanowane, a stare komunikaty i zmienione strony są zapisywane na dysku. Jeśli punkty kontrolne są zbyt częste, może to mieć wpływ na wydajność. Wartość parametru LOGLOAD może mieć również wpływ na częstotliwość punktów kontrolnych. Jeśli menedżer kolejek zostanie nieprawidłowo zakończony, to przy restarcie może być konieczne odczytanie z powrotem do 3 punktów kontrolnych. Najlepszym odstępem między punktami kontrolnymi jest równowaga między działaniem w momencie wykonania punktu kontrolnego, a ilością danych dziennika, które mogą wymagać odczytu podczas restartowania menedżera kolejek.

Istnieje znaczny narzut związany z uruchomieniem kanału. Zwykle lepiej jest uruchomić kanał i pozostawić go połączone, a nie częste starty i przystanki kanału.

Informacje pokrewne

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

z/OS Aplikacje pomostowe IMS i IMS w systemie IBM MQ for z/OS

Te informacje ułatwiają pisanie aplikacji produktu IMS przy użyciu produktu IBM MQ.

- Informacje na temat używania punktów synchronizacji i wywołań MQI w aplikacjach IMS zawiera sekcja “Tworzenie aplikacji produktu IMS przy użyciu produktu IBM MQ” na stronie 71.
- Aby napisać aplikacje, które korzystają z mostu IBM MQ - IMS , należy zapoznać się z “Pisanie aplikacji mostu IMS” na stronie 75.

Użyj poniższych odsyłaczy, aby dowiedzieć się więcej na temat aplikacji mostu IMS i IMS w systemie IBM MQ for z/OS:

- “Tworzenie aplikacji produktu IMS przy użyciu produktu IBM MQ” na stronie 71
- “Pisanie aplikacji mostu IMS” na stronie 75

Pojęcia pokrewne

“Interfejs kolejki komunikatów-przegląd” na stronie 811
Informacje na temat komponentów interfejsu kolejek komunikatów (Message Queue Interface-MQI).

“Nawiąże połączenie z menedżerem kolejek i odłączenie go od niego” na stronie 824
Aby można było korzystać z usług programistycznych produktu IBM MQ , program musi mieć połączenie z menedżerem kolejek. Informacje zawarte w tej sekcji umożliwiają zapoznanie się z informacjami na temat nawiązywania połączenia z menedżerem kolejek i z jego rozłączeniem.

“Otwieranie i zamykanie obiektów” na stronie 833
Te informacje udostępniają wgląd w otwieranie i zamykanie obiektów produktu IBM MQ .

“Umieszczanie komunikatów w kolejce” na stronie 844
Te informacje umożliwiają zapoznanie się z informacjami na temat umieszczania komunikatów w kolejce.

“Pobieranie komunikatów z kolejki” na stronie 860
Ta sekcja zawiera informacje na temat pobierania komunikatów z kolejki.

“Sprawdzanie i ustawianie atrybutów obiektu” na stronie 943
Atrybuty to właściwości, które definiują parametry obiektu IBM MQ .

“Zatwierdzanie i wycofywanie jednostek pracy” na stronie 946
W tej sekcji opisano sposób zatwierdzania i tworzenia kopii zapasowych wszystkich możliwych do odtworzenia operacji get i put, które wystąpiły w jednostce pracy.

“Uruchamianie aplikacji produktu IBM MQ przy użyciu wyzwalaczy” na stronie 958
Informacje na temat wyzwalaczy i sposobów uruchamiania aplikacji IBM MQ przy użyciu wyzwalaczy.

“Praca z interfejsem MQI i klastrami” na stronie 978
Istnieją specjalne opcje dotyczące wywołań i kodów powrotu, które odnoszą się do technologii klastrowej.

“Używanie i zapisywanie aplikacji w systemie IBM MQ for z/OS” na stronie 983
Aplikacje produktu IBM MQ for z/OS mogą być uruchamiane z programów działających w wielu różnych środowiskach. Oznacza to, że mogą skorzystać z udogodnień dostępnych w więcej niż jednym środowisku.

Tworzenie aplikacji produktu IMS przy użyciu produktu IBM MQ

Podczas korzystania z produktu IBM MQ w aplikacjach IMS należy wziąć pod uwagę dodatkowe uwagi dotyczące tego, które wywołania funkcji API produktu MQ mogą być używane, a także mechanizm używany w punkcie synchronizacji.

Użyj poniższych odsyłaczy, aby dowiedzieć się więcej na temat pisania aplikacji IMS w systemie IBM MQ for z/OS:

- “Punkty synchronizacji w aplikacjach IMS” na stronie 71
- “Wywołania MQI w aplikacjach IMS” na stronie 72

Ograniczenia

Istnieją ograniczenia, których wywołania funkcji API produktu IBM MQ mogą być używane przez aplikację przy użyciu adaptera IMS .

Następujące wywołania funkcji API produktu IBM MQ nie są obsługiwane w ramach aplikacji przy użyciu adaptera IMS :

- MQCB
- MQCB_FUNCTION
- Komenda MQCTL

Pojęcia pokrewne

“Pisanie aplikacji mostu IMS” na stronie 75
Ten temat zawiera informacje na temat pisania aplikacji do korzystania z mostu IBM MQ - IMS .

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.

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 72
- “Aplikacje do uzyskiwania informacji” na stronie 74

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
```

Pisanie aplikacji mostu IMS

Ten temat zawiera informacje na temat pisania aplikacji do korzystania z mostu IBM MQ - IMS .

Informacje na temat mostu IBM MQ - IMS znajdują się w sekcji Most IMS.

Użyj poniższych odsyłaczy, aby dowiedzieć się więcej na temat pisania aplikacji mostu IMS w systemie IBM MQ for z/OS:

- “Sposób, w jaki most IMS obsługuje komunikaty” na stronie 75
- “Pisanie programów transakcyjnych IMS za pomocą programu IBM MQ” na stronie 1006

Pojęcia pokrewne

“Tworzenie aplikacji produktu IMS przy użyciu produktu IBM MQ” na stronie 71
Podczas korzystania z produktu IBM MQ w aplikacjach IMS należy wziąć pod uwagę dodatkowe uwagi dotyczące tego, które wywołania funkcji API produktu MQ mogą być używane, a także mechanizm używany w punkcie synchronizacji.

Sposób, w jaki most IMS obsługuje komunikaty

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 produktu IBM MQ , które zostały zdefiniowane z klasą pamięci masowej, która określa grupę XCF i nazwę elementu docelowego systemu IMS . Są one nazywane kolejkami mostów MQ-IMS lub po prostu kolejkami **bridge** .

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

Przed wysłaniem komunikatów do aplikacji IMS użytkownik nie musi logować się do produktu IMS . Identyfikator użytkownika w polu *UserIdentifier* struktury MQMD jest używany do sprawdzania zabezpieczeń. Poziom sprawdzania jest określany, gdy program IBM MQ łączy się z serwerem IMSi jest opisany w sekcji Kontrola dostępu do aplikacji dla mostu IMS. Umożliwia to zaimplementowanie pseudo signon.

Most IBM MQ - IMS akceptuje następujące typy komunikatów:

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

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

Uwaga:

1. Nawiasy kwadratowe, [], reprezentują opcjonalne wielosegmenty.
2. Ustaw pole *Format* struktury MQMD na wartość MQFMT_IMS w celu użycia 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 plus długość wartości MQIIH (jeśli jest obecna) jest równa długości komunikatu.

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

- Jeśli komunikat zawiera strukturę MQIIH, most weryfikuje wartość 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 za pomocą komunikatu DFS1288E . Jeśli kod transakcji reprezentuje komendę, program IMS wykonuje komendę; w przeciwnym razie komunikat znajduje się w kolejce IMS dla transakcji.
- Jeśli komunikat zawiera dane transakcji IMS , ale nie ma struktury MQIIH, most IMS tworzy 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 (zatwierdź-następnie-wyślij)
  - *Format* w strukturze MQMD jest używany jako *MFSMapName* (na wejściu)
  - Tryb zabezpieczeń to MQISS_CHECK

  Komunikat odpowiedzi jest również zbudowany bez struktury MQIIH, przyjmując *Format* dla deskryptora MQMD z *MFSMapName* danych wyjściowych IMS .

Most IBM MQ - IMS korzysta z jednej lub dwóch potoków dla każdej kolejki IBM MQ :

- Zsynchronizowana rura jest używana dla wszystkich komunikatów przy użyciu trybu zatwierdzania 0 (COMMIT_THEN_SEND) (te pokazy ze SYN w polu statusu komendy IMS /DIS TMEMBER klienta TPIPE xxxx)
- Niezsynchronizowana Tpipe jest używana dla wszystkich komunikatów przy użyciu trybu kontroli transakcji 1 (SEND_THEN_COMMIT).

Potoki są tworzone przez produkt IBM MQ , gdy są one używane jako pierwsze. Niezsynchronizowana potok Tpipe istnieje do momentu zrestartowania serwera IMS . Zsynchronizowane trury istnieją do momentu, gdy program IMS nie jest zimny. Nie można samodzielnie usunąć tych potoków.

Więcej informacji na temat sposobu obsługi mostu IBM MQ - IMS z komunikatami można znaleźć w następujących tematach:

- “Odwzorowanie komunikatów IBM MQ na typy transakcji IMS” na stronie 77
- “Jeśli komunikat nie może zostać umieszczony w kolejce produktu IMS” na stronie 77
- “Kody sprzężenia zwrotnego mostu IMS” na stronie 78
- “Pola MQMD w komunikatach z mostu IMS” na stronie 78
- “Pola MQIIH w komunikatach z mostu IMS” na stronie 79
- “Komunikaty odpowiedzi z programu IMS” na stronie 80
- “Korzystanie z alternatywnych PCB odpowiedzi w transakcjach IMS” na stronie 81
- “Wysyłanie niezamówionych komunikatów z produktu IMS” na stronie 81
- “Segmentacja komunikatów” na stronie 81
- “Konwersja danych dla komunikatów do i z mostu IMS” na stronie 81

Pojęcia pokrewne

“Pisanie programów transakcyjnych IMS za pomocą programu IBM MQ” na stronie 1006
Kodowanie wymagane do obsługi transakcji IMS za pomocą produktu IBM MQ zależy od formatu komunikatu wymaganego przez transakcję IMS oraz zakresu odpowiedzi, które może zwrócić. Jednak w sytuacji, gdy aplikacja obsługuje informacje o formatowaniu ekranu produktu IMS , należy wziąć pod uwagę kilka punktów.

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 • Transakcje nienaprawialne są odrzucane przez IMS	• Transakcje krótkiej ścieżki • Transakcje konwersacyjne • W pełni funkcjonalne transakcje
Nietrwałe komunikaty IBM MQ	• Nienaprawialne transakcje z pełnymi funkcjami • Transakcje odtwarzalne są dozwolone w poprawkach IMS V8 i APAR PQ61404 oraz we wszystkich późniejszych wersjach systemu IMS .	• Transakcje krótkiej ścieżki • Transakcje konwersacyjne • 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) .

Jeśli komunikat nie może zostać umieszczony w kolejce produktu IMS

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

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

- Jeśli komunikat nie może zostać umieszczony w programie IMS , ponieważ komunikat jest niepoprawny, komunikat jest umieszczany w kolejce niedostarczonych komunikatów, a do konsoli systemowej wysyłany jest komunikat.
- 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 komunikat IBM MQ zawierający przyczynę niepowodzenia w kolejce odpowiedzi.

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

Aby ponownie wysłać komunikaty, należy wykonać ***jeden*** z następujących działań:

- Zatrzymaj i zrestartuj potoki w programie IMS , które odpowiadają kolejce
- Zmień kolejkę na GET (WYŁĄCZONE), a następnie ponownie w celu uzyskania (ENABLED)
- Zatrzymaj i zrestartuj produkt IMS lub OTMA
- Zatrzymaj i zrestartuj podsystem IBM MQ .

- Jeśli komunikat został odrzucony przez produkt IMS w przypadku innych niż błąd komunikatów, komunikat IBM MQ zostanie zwrócony do kolejki źródłowej, program IBM MQ zatrzyma przetwarzanie kolejki, a do konsoli systemowej zostanie wysłany komunikat o błędzie.

  Jeśli wymagany jest komunikat z raportem o wyjątku, most umieszcza go w kolejce odpowiedzi z uprawnieniami inicjatora. Jeśli komunikat nie może zostać umieszczony w kolejce, komunikat raportu jest umieszczany w kolejce niedostarczonych komunikatów z uprawnieniami mostu. Jeśli nie można go umieścić w kolejce DLQ, zostanie ona odrzucona.

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.

Pola MQMD w komunikatach z mostu IMS
Informacje na temat pól MQMD w komunikatach z mostu IMS .

Kod MQMD komunikatu źródłowego jest przenoszony przez produkt IMS w sekcji Dane użytkownika w nagłówkach OTMA. Jeśli komunikat pochodzi z produktu IMS, jest on budowany przy użyciu wyjścia rozstrzygania miejsca docelowego produktu IMS . Kod MQMD komunikatu odebranego z produktu IMS jest zbudowany w następujący sposób:

StrucID
"MD"

Wersja
MQMD_VERSION_1

Raport
MQRO_NONE

MsgType
MQMT_REPLY

Utrata ważności
Jeśli wartość MQIIH_PASS_EXPIRATION jest ustawiona w polu flagi w tabeli MQIIH, to pole zawiera pozostały czas utraty ważności, w przeciwnym razie jest on ustawiony na wartość MQEI_UNLIMITED.

Opinie
MQFB_NONE

Kodowanie
MQENC.Native (kodowanie systemu z/OS)

CodedCharSetId
MQCCSI_Q_MGR (CodedCharSetID w systemie 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 kontroli transakcji: 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 (ustawiane na lokalną nazwę 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 ma błędu, w przeciwnym razie nazwa QMGR

PutDate
Data umieszczenia komunikatu

PutTime
Czas umieszczenia komunikatu

Dane_pochodzenia_aplikacji
Puste

Pola MQIIH w komunikatach z mostu IMS
Informacje na temat pól MQIIH w komunikatach z mostu IMS .

Obiekt MQIIH komunikatu odebranego z produktu IMS jest zbudowany w następujący sposób:

StrucId
"IIH"

Wersja
1

StrucLength
84

Kodowanie
MQENC_NATIVE

CodedCharSetId
MQCCSI_Q_MGR

Format
MQIIH.ReplyToFormat komunikatu wejściowego, jeśli MQIIH.ReplyToFormat nie jest pusta, w przeciwnym razie IOPCB.MODNAME

Flagi
0

LTermOverride
Nazwa LTERM (Tpipe) z nagłówka OTMA

MFSMapName
Nazwa odwzorowania z nagłówka OTMA

Format ReplyTo
Puste

Authenticator
MQIIH.Authenticator komunikatu wejściowego, jeśli komunikat odpowiedzi jest umieszczany w kolejce mostu MQ-IMS . W przeciwnym razie puste pole jest puste.

Identyfikator TranInstance
Identyfikator konwersacji/znacznik serwera z nagłówka OTMA, jeśli konwersacja jest konwersacja. W wersjach systemu IMS wcześniejszych niż V14, to pole zawsze jest puste, jeśli nie jest w konwersacji. Począwszy od systemu IMS V14 , pole to może być ustawione przez system IMS , nawet jeśli nie 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

Komunikaty odpowiedzi z programu IMS
Gdy transakcja IMS jest transakcją ISRTs z jej IOPCB, komunikat jest kierowany z powrotem do źródłowego LTERM lub TPIPE.

Są one widoczne w produkcie IBM MQ jako komunikaty odpowiedzi. Komunikaty odpowiedzi z programu IMS są umieszczane w kolejce odpowiedzi określonej w pierwotnym komunikacie. Jeśli komunikat nie może zostać umieszczony 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 programu IMS zostanie wysłane potwierdzenie negatywne, co oznacza, że komunikat nie może zostać odebrany. Odpowiedzialność za komunikat jest następnie zwracana do programu IMS. Jeśli używany jest tryb kontroli transakcji 0, komunikaty z tego protokołu Tpipe nie są wysyłane do mostu i pozostają w kolejce produktu IMS . To znaczy, że kolejne komunikaty nie są wysyłane do czasu restartu. Jeśli używany jest tryb kontroli transakcji 1, inne prace mogą być kontynuowane.

Jeśli odpowiedź ma strukturę MQIIH, jej typem formatu jest MQFMT_IMS;, jeśli nie, jego typ formatu jest określany przez nazwę MOD produktu IMS używaną podczas wstawiania komunikatu.

Korzystanie z alternatywnych PCB odpowiedzi w transakcjach IMS
Gdy transakcja IMS korzysta z alternatywnych PCB odpowiedzi (ISRTs do ALTPCB lub wysyła wywołanie CHNG do modyfikowalnego bloku PCB), wywołanie wyjścia routingu wstępnego (DFSYPRX0) jest wywoływane w celu określenia, czy komunikat powinien zostać ponownie uruchomiony.

Jeśli komunikat ma zostać przekierowany, następuje wywołanie wyjścia rozstrzygania miejsca docelowego (DFSYDRU0) w celu potwierdzenia miejsca docelowego i przygotowania informacji o nagłówku. Patrz sekcja Korzystanie z wyjść OTMA w programie IMS i Wyjście z routingu wstępnego DFSYPRX0 , aby uzyskać informacje na temat tych programów obsługi wyjścia.

Jeśli działanie nie zostanie wykonane w wyjściach, wszystkie dane wyjściowe z IMS transakcji zainicjowanych z menedżera kolejek produktu IBM MQ , niezależnie od tego, czy mają być wykonywane przez IOPCB, czy też do grupy ALTPCB, zostaną zwrócone do tego samego menedżera kolejek.

Wysyłanie niezamówionych komunikatów z produktu IMS
Aby wysyłać komunikaty z produktu IMS do kolejki produktu IBM MQ , należy wywołać transakcję IMS , która jest używana przez ISRTs do bazy danych ALTPCB.

Aby kierować niezamówionym komunikatami z produktu IMS i zbudować dane użytkownika OTMA, należy napisać procedury poprzedzające routing i rozwiązywanie miejsca docelowego, tak aby można było poprawnie zbudować deskryptor MQMD komunikatu. Informacje na temat tych programów obsługi wyjścia można znaleźć w sekcji Wyjście routingu wstępnego DFSYPRX0 i Wyjście użytkownika rozstrzygania miejsca docelowego .

Uwaga: Most IBM MQ - IMS nie wie, czy komunikat, który otrzymuje, jest odpowiedzią, czy niezamówionym komunikatem. Obsługuje on ten sam komunikat w każdym przypadku, budując tabelę MQMD i MQIIH odpowiedzi na podstawie UserData OTMA, które dotarło do komunikatu.

Niezamówione komunikaty mogą tworzyć nowe potoków. Na przykład, jeśli istniejąca transakcja IMS została przełączona na nowy LTERM (na przykład PRINT01), ale implementacja wymaga, aby dane wyjściowe były dostarczane za pośrednictwem OTMA, zostanie utworzona nowa Tpipe (w tym przykładzie o nazwie PRINT01). Domyślnie jest to Tpipe, który nie jest zsynchronizowany. Jeśli implementacja wymaga, aby komunikat mógł być odtwarzalny, należy ustawić flagę wyjścia wyjścia rozstrzygania miejsca docelowego. Więcej informacji na ten temat zawiera podręcznik *IMS Customization Guide* .

Segmentacja komunikatów
Transakcje IMS można zdefiniować jako oczekiwane dane wejściowe jedno-lub wielosegmentowe.

Źródłowa aplikacja IBM MQ musi konstruować dane wejściowe użytkownika po strukturze MQIIH jako co najmniej jeden segment danych LLZZ. Wszystkie segmenty komunikatu IMS muszą być zawarte w pojedynczym komunikacie IBM MQ wysłanym z pojedynczą MQPUT.

Maksymalna długość segmentu danych LLZZ jest definiowana przez system IMS/OTMA (32767 bajtów). Całkowita długość komunikatu IBM MQ jest sumą bajtów LL oraz długością struktury MQIIH.

Wszystkie segmenty odpowiedzi są zawarte w jednym komunikacie IBM MQ .

Istnieje dalsze ograniczenie dotyczące ograniczenia 32 kB dla komunikatów o formacie MQFMT_IMS_VAR_STRING. Gdy dane w komunikacie ASCII o mieszanym identyfikatorze CCSID są konwertowane na komunikat o kodowaniu EBCDIC mieszanym, bajt shift-in lub shift-out jest dodawany za każdym razem, gdy istnieje przejście między znakami SBCS i DBCS. Ograniczenie o wielkości 32 kB ma zastosowanie do maksymalnej wielkości komunikatu. Oznacza to, że ponieważ pole LL w komunikacie nie może przekroczyć 32 kB, komunikat nie może być większy niż 32 kB, w tym wszystkie znaki shift-in i shift-out. Aplikacja budowania komunikatu musi zezwalać na to działanie.

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 zawierającej informacje XCF zdefiniowane dla

swojej klasy pamięci masowej. Konwersja danych nie jest wykonywana, gdy komunikat jest dostarczany do kolejki publikowania/subskrybowania.

Wszystkie wymagane wyjścia muszą być dostępne dla rozproszonego narzędzia kolejkowania w zestawie danych przywoływanym przez instrukcję CSQXLIB DD. Oznacza to, że można wysyłać komunikaty do aplikacji IMS , korzystając z mostu IBM MQ - IMS z dowolnej platformy IBM MQ .

Jeśli wystąpiły błędy konwersji, komunikat jest umieszczany w kolejce bez konwersji. W rezultacie w końcu jest on traktowany jako błąd przez most IBM MQ - IMS , ponieważ most nie może rozpoznać formatu nagłówka. Jeśli wystąpi błąd konwersji, do konsoli z/OS zostanie wysłany komunikat o błędzie.

Szczegółowe informacje na temat konwersji danych można znaleźć w sekcji “Pisanie wyjść konwersji danych” na stronie 1082 .

Wysyłanie komunikatów do mostu IBM MQ - IMS

Aby upewnić się, że konwersja jest wykonywana poprawnie, należy poinformować menedżera kolejek o tym, jaki jest format komunikatu.

Jeśli komunikat ma strukturę MQIIH, wartość *Format* w strukturze MQMD musi być ustawiona na wbudowany format MQFMT_IMS, a wartość *Format* w tabeli MQIIH musi być ustawiona na nazwę formatu opisowego, który opisuje dane komunikatu. Jeśli nie ma tabeli MQIIH, ustaw wartość *Format* w strukturze MQMD na nazwę formatu.

Jeśli dane (inne niż LLZZs) to wszystkie dane znakowe (MQCHAR), należy użyć jako nazwy formatu (w tabeli MQIIH lub MQMD, w zależności od przypadku), wbudowanego formatu MQFMT_IMS_VAR_STRING. W przeciwnym razie należy użyć własnej nazwy formatu, w którym to przypadku należy również podać wyjście konwersji danych dla formatu. Wyjście musi obsługiwać konwersję LLZZs w swoim komunikacie, oprócz samych danych (ale nie musi obsługiwać żadnej MQIIH na początku wiadomości).

Jeśli aplikacja używa produktu *MFSMapName*, zamiast tego można użyć komunikatów z produktem MQFMT_IMS i zdefiniować nazwę odwzorowania przekazanej do transakcji IMS w polu MFSMapName tabeli MQIIH.

Odbieranie komunikatów z mostu IBM MQ - IMS

Jeśli w oryginalnym komunikacie, który jest wysyłany do programu IMS, znajduje się struktura MQIIH, jest ona również obecna w komunikacie odpowiedzi.

Aby upewnić się, że odpowiedź została poprawnie przekształcona:

- Jeśli w pierwotnym komunikacie znajduje się struktura MQIIH, w polu MQIIH *ReplytoFormat* oryginalnego komunikatu należy określić format, który ma być wyświetlany dla komunikatu odpowiedzi. Ta wartość jest umieszczana w polu MQIIH *Format* komunikatu odpowiedzi. Jest to szczególnie przydatne w przypadku, gdy wszystkie dane wyjściowe mają postać LLZZ < dane znakowe >.
- Jeśli w oryginalnym komunikacie nie ma struktury MQIIH, określ format, który ma być wyświetlany dla komunikatu odpowiedzi jako nazwa MFS MOD w ISRT aplikacji IMS do IOPCB.

Tworzenie aplikacji JMS i Java

Produkt IBM MQ udostępnia dwa interfejsy języka Java : IBM MQ classes for Java Message Service i IBM MQ classes for Java.

O tym zadaniu

W produkcie IBM MQ istnieją dwa alternatywne interfejsy API do użycia w aplikacjach Java . A Java application can use either IBM MQ classes for JMS or IBM MQ classes for Java to access IBM MQ resources.

IBM MQ classes for JMS
IBM MQ classes for Java Message Service (JMS) jest dostawcą JMS dostarczonym z produktem IBM MQ. 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.

Jeśli użytkownik nie zna produktu IBM MQlub ma już doświadczenie w produkcie JMS , może okazać się łatwiejszy w użyciu znanego interfejsu API produktu JMS w celu uzyskania dostępu do zasobów produktu IBM MQ przy użyciu produktu IBM MQ classes for JMS. JMS jest również integralną częścią platformy Java Platform, Enterprise Edition (Java EE). Aplikacje produktu Java EE mogą używać komponentów bean sterowanych komunikatami (message-driven bean-MDB) do asynchronicznego przetwarzania komunikatów. Produkt JMS jest również standardowym mechanizmem dla produktu Java EE do interakcji z asynchronicznymi systemami przesyłania komunikatów, takimi jak IBM MQ. Każdy serwer aplikacji, który jest zgodny z produktem Java EE , musi zawierać dostawcę JMS . W związku z tym produkt JMS może być używany do komunikacji między różnymi serwerami aplikacji lub do portu aplikacji z jednego dostawcy JMS do innego bez żadnych zmian w aplikacji.

IBM MQ classes for Java

Produkt IBM MQ classes for Java umożliwia korzystanie z produktu IBM MQ w środowisku produktu Java . Program IBM MQ classes for Java umożliwia aplikacji Java łączenie się z serwerem IBM MQ jako klient IBM MQ lub bezpośrednie połączenie z menedżerem kolejek produktu IBM MQ .

IBM MQ classes for Java encapsulates the Message Queue Interface (MQI), the native IBM MQ API, and uses the same object model as other object-oriented interfaces, whereas IBM MQ classes for Java Message Service implements Oracle's Java Message Service (JMS) interfaces.

Jeśli użytkownik jest zaznajomiony z produktem IBM MQ w środowiskach innych niż Java, korzystając z języków proceduralnych lub obiektowych, można przenieść istniejącą wiedzę do środowiska Java za pomocą programu IBM MQ classes for Java. Można również wykorzystać pełny zakres funkcji produktu IBM MQ, z których nie wszystkie są dostępne w produkcie IBM MQ classes for JMS.

Uwaga:

Produkt IBM nie rozszerzy IBM MQ classes for Java i nie będzie on funkcjonalnie stabilizowany na poziomie dostarczonym w produkcie IBM MQ 8.0. Istniejące aplikacje, które korzystają z produktu IBM MQ classes for Java , nadal są w pełni obsługiwane, ale nowe funkcje nie zostaną dodane, a żądania dotyczące rozszerzeń zostaną odrzucone. W pełni obsługiwane oznacza, że defekty zostaną naprawione wraz ze zmianami wymaganowymi przez zmiany w wymaganiach systemowych produktu IBM MQ .

IBM MQ classes for Java nie są obsługiwane w produkcie IMS.

IBM MQ classes for Java nie są obsługiwane w produkcie WebSphere Liberty. Nie mogą one być używane z funkcją przesyłania komunikatów IBM MQ Liberty ani z ogólną obsługą produktu JCA . Więcej informacji na ten temat zawiera sekcja Korzystanie z interfejsów Java produktu WebSphere MQ w środowiskach J2EE/JEE.

użycieIBM MQ classes for JMS

IBM MQ classes for Java Message Service (IBM MQ classes for JMS) jest dostawcą JMS dostarczonym z produktem IBM MQ. Oprócz implementowania interfejsów zdefiniowanych w pakiecie javax.jms produkt IBM MQ classes for JMS udostępnia dwa zestawy rozszerzeń do interfejsu API produktu JMS .

Specyfikacja JMS definiuje zestaw interfejsów, których aplikacje mogą używać do wykonywania operacji przesyłania komunikatów. Najnowsza wersja specyfikacji to JMS 2.0. Pakiet javax.jms definiuje interfejsy produktu JMS , a dostawca produktu JMS implementuje te interfejsy dla konkretnego produktu przesyłania komunikatów. IBM MQ classes for JMS jest dostawcą JMS , który implementuje interfejsy JMS dla produktu IBM MQ.

Specyfikacja JMS oczekuje, że obiekty ConnectionFactory i Destination mają być administrowane obiektami. Administrator tworzy i obsługuje administrowane obiekty w centralnym repozytorium, a aplikacja JMS pobiera te obiekty za pomocą Java Naming Directory Interface (JNDI). Produkt IBM MQ classes for JMS obsługuje użycie administrowanych obiektów, a administrator może użyć narzędzia administracyjnego IBM MQ JMS lub produktu IBM MQ Explorer w celu utworzenia i obsługi obiektów administrowanych.

Produkt IBM MQ classes for JMS udostępnia także dwa zestawy rozszerzeń do interfejsu API produktu JMS . Główne uwagi dotyczące tych rozszerzeń dotyczą tworzenia i konfigurowania fabryk połączeń

i miejsc docelowych dynamicznie w czasie wykonywania, ale rozszerzenia udostępniają również funkcje, które nie są bezpośrednio związane z przesyłaniem komunikatów, takie jak funkcja określania problemu.

Rozszerzenia IBM MQ JMS
Poprzednie wersje produktu IBM MQ classes for JMS zawierają rozszerzenia implementowane w obiektach, takich jak obiekty MQConnectionFactory, MQQueue i MQTopic. Te obiekty mają właściwości i metody specyficzne dla produktu IBM MQ. Obiektami mogą być obiekty administrowane, a aplikacja może tworzyć obiekty dynamicznie w czasie wykonywania. Ta wersja produktu IBM MQ classes for JMS obsługuje te rozszerzenia, które są teraz nazywane rozszerzeniami IBM MQ JMS . Można nadal używać, bez zmian, żadnych aplikacji, które używają tych rozszerzeń.

Rozszerzenia IBM JMS
W tej wersji produktu IBM MQ classes for JMS udostępniono bardziej ogólny zestaw rozszerzeń do interfejsu API produktu JMS , które nie są specyficzne dla produktu IBM MQ jako systemu przesyłania komunikatów. Rozszerzenia te są znane jako rozszerzenia IBM JMS i mają następujące ogólne cele:

- Zapewnienie większego poziomu spójności dla dostawców IBM JMS
- Aby ułatwić zapis aplikacji pomostowej między dwoma systemami przesyłania komunikatów produktu IBM
- Aby ułatwić sobie port aplikacji z jednego dostawcy IBM JMS do innego

Rozszerzenia udostępniają funkcje, które są podobne do funkcji udostępnionych w produkcie IBM Message Service Client for C/C++ i IBM Message Service Client for .NET.

W produkcie IBM MQ 8.0produkt IBM MQ classes for JMS jest budowany z produktem Java 7.

Środowisko wykonawcze produktu Java 7 obsługuje uruchamianie wcześniejszych wersji plików klas.

Pojęcia pokrewne
Interfejsy językowe programu IBM MQ Java
“Model JMS” na stronie 134
Model JMS definiuje zestaw interfejsów, które mogą być używane przez aplikacje produktu Java do wykonywania operacji przesyłania komunikatów. IBM MQ classes for JMS, jako dostawca JMS , definiuje sposób, w jaki obiekty JMS są powiązane z pojęciami IBM MQ . Specyfikacja JMS oczekuje, że niektóre obiekty JMS będą administrowane obiektami. Produkt JMS 2.0 wprowadza uproszczony interfejs API, zachowując jednocześnie klasyczny interfejs API z produktu JMS 1.1.

“Korzystanie z funkcji produktu JMS 2.0” na stronie 319
Produkt JMS 2.0 wprowadza kilka nowych obszarów funkcjonalności do produktu IBM MQ classes for JMS.

Dlaczego należy używać produktu IBM MQ classes for JMS?

Korzystanie z produktu IBM MQ classes for JMS ma wiele zalet, w tym możliwość ponownego wykorzystania wszystkich istniejących umiejętności JMS w organizacji, a aplikacje są bardziej niezależne od dostawcy JMS i podstawowej konfiguracji produktu IBM MQ .

Produkt IBM MQ classes for JMS jest jednym z dwóch alternatywnych interfejsów API, które mogą być używane przez aplikacje produktu Java w celu uzyskania dostępu do zasobów produktu IBM MQ . Innym interfejsem API jest IBM MQ classes for Java. Mimo że istniejące aplikacje, które korzystają z produktu IBM MQ classes for Java , nadal są w pełni obsługiwane, nowe aplikacje powinny używać IBM MQ classes for JMS (patrz “Wybór funkcji API” na stronie 86).

Podsumowanie zalet korzystania z produktu IBM MQ classes for JMS

Użycie produktu IBM MQ classes for JMS umożliwia ponowne wykorzystanie istniejących umiejętności JMS i zapewnienie niezależności aplikacji.

- Możliwe jest ponowne wykorzystanie umiejętności produktu JMS .

  IBM MQ classes for JMS jest dostawcą JMS , który implementuje interfejsy JMS dla systemu IBM MQ jako system przesyłania komunikatów. Jeśli dana organizacja jest nowa w produkcie IBM MQ, ale już ma umiejętność programowania aplikacji w wersji JMS , łatwiej jest użyć znanego interfejsu API produktu

JMS w celu uzyskania dostępu do zasobów produktu IBM MQ , a nie do jednego z innych interfejsów API udostępnionych z produktem IBM MQ.

- JMS jest integralną częścią produktu Java Platform, Enterprise Edition (Java EE).

  JMS jest naturalnym interfejsem API, który ma być używany do przesyłania komunikatów na platformie Java EE . Każdy serwer aplikacji, który jest zgodny z produktem Java EE , musi zawierać dostawcę JMS . Produktu JMS można używać w klientach aplikacji, serwletach, stronach serwera Java (JSP), komponentach EJB (Enterprise Java Bean) oraz komponentach bean sterowanych komunikatami (message driven bean-MDB). Należy zwrócić uwagę, że aplikacje produktu Java EE korzystają z komponentów MDB w celu asynchronicznego przetwarzania komunikatów, a wszystkie komunikaty są dostarczane do komponentów MDB jako komunikaty produktu JMS .

- Fabryki połączeń i miejsca docelowe mogą być przechowywane jako obiekty administrowane w produkcie JMS w centralnym repozytorium, a nie są zakodowane na stałe w aplikacji.

  Administrator może tworzyć i obsługiwać obiekty administrowane JMS w centralnym repozytorium, a aplikacje produktu IBM MQ classes for JMS mogą pobierać te obiekty za pomocą Java Naming Directory Interface (JNDI). Fabryki połączeń i miejsca docelowe produktu JMS hermetyują 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, to informacje te nie są zakodowane na stałe w aplikacji. W związku z tym układ ten udostępnia aplikację o stopniu niezależności od bazowej konfiguracji produktu IBM MQ .

- JMS to standardowy interfejs API, który może zapewnić przenośność aplikacji.

  Aplikacja JMS może używać produktu JNDI do pobierania fabryk połączeń i miejsc docelowych, które są przechowywane jako obiekty administrowane, i używać tylko interfejsów zdefiniowanych w pakiecie javax.jms w celu wykonania operacji przesyłania komunikatów. Aplikacja jest wtedy całkowicie niezależna od dowolnego dostawcy JMS , takiego jak IBM MQ classes for JMS, i może być importowana z jednego dostawcy JMS do innego bez żadnych zmian w aplikacji. Jeśli produkt JNDI nie jest dostępny w określonym środowisku aplikacji, aplikacja IBM MQ classes for JMS może używać rozszerzeń do interfejsu API produktu JMS w celu tworzenia i konfigurowania fabryk połączeń i miejsc docelowych dynamicznie w czasie wykonywania. Aplikacja jest wtedy całkowicie samodzielna, ale jest powiązana z produktem IBM MQ classes for JMS jako dostawcą JMS .

- Aplikacje pomostowe mogą być łatwiejsze do pisania przy użyciu produktu 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 przy użyciu specyficznych dla produktu interfejsów API i formatów komunikatów. Zamiast tego można napisać aplikację pomostową, korzystając z dwóch dostawców JMS , po jednym dla każdego systemu przesyłania komunikatów. Następnie aplikacja używa tylko jednego interfejsu API, interfejsu API JMS i przetwarza tylko komunikaty produktu JMS .

Środowiska do wdrożenia

Aby zapewnić integrację z serwerem aplikacji Java EE , standardy produktu Java EE wymagają, aby dostawcy przesyłania komunikatów dostarczali adapter zasobów. Po specyfikacji Java EE Connector Architecture (JCA) produkt IBM MQ udostępnia adapter zasobów, który używa produktu JMS do udostępniania funkcji przesyłania komunikatów w dowolnym certyfikowanym środowisku produktu Java EE .

Chociaż możliwe było użycie IBM MQ classes for Java wewnątrz Java EE, ta funkcja API nie jest zaprojektowana ani zoptymalizowana w tym celu. Należy zapoznać się z notą techniczną IBM Korzystanie z interfejsów Java produktu WebSphere MQ w środowiskach J2EE/JEE , aby uzyskać szczegółowe informacje na temat zagadnień związanych z produktem IBM MQ classes for Java w produkcie Java EE.

Poza środowiskiem produktu Java EE udostępniane są pliki OSGi i JAR, co ułatwia uzyskanie tylko produktu IBM MQ classes for JMS. Te pliki JAR są teraz bardziej łatwo wdrażalne albo autonomiczne, albo w ramach środowisk zarządzania oprogramowaniem, takich jak Maven. Więcej informacji na ten temat zawiera nota techniczna IBM Uzyskiwanie klas produktu WebSphere MQ dla usługi JMS.

Wybór funkcji API

Nowe aplikacje powinny używać IBM MQ classes for JMS zamiast IBM MQ classes for Java.

Produkt IBM MQ classes for JMS zapewnia dostęp zarówno do funkcji przesyłania komunikatów w trybie punkt z punktem, jak i do publikowania/subskrypcji produktu IBM MQ. Oprócz wysyłania komunikatów produktu JMS , które zapewniają obsługę standardowego modelu przesyłania komunikatów produktu JMS , aplikacje mogą również wysyłać i odbierać komunikaty bez dodatkowych nagłówków, a więc mogą współpracować z innymi aplikacjami produktu IBM MQ , na przykład aplikacjami środowiska C MQI. Dostępne są pełne sterowanie ładowaniem komunikatów MQMD i MQ . Dostępne są również dodatkowe funkcje produktu IBM MQ , takie jak strumieniowanie komunikatów, asynchroniczne komunikaty put i raporty. Za pomocą dostarczonych klas programu pomocniczego PCF komunikaty administracyjne programu IBM MQ PCF mogą być wysyłane i odbierane za pośrednictwem interfejsu API produktu JMS i mogą być używane do administrowania menedżerami kolejek.

Funkcje, które ostatnio zostały dodane do produktu IBM MQ, takie jak asynchroniczne zużywanie i automatyczne ponowne połączenie, nie są dostępne w IBM MQ classes for Java, ale są dostępne w IBM MQ classes for JMS. Istniejące aplikacje, które korzystają z produktu IBM MQ classes for Java , nadal są w pełni obsługiwane.

Jeśli wymagany jest dostęp do składników produktu IBM MQ , które nie są dostępne za pośrednictwem produktu IBM MQ classes for JMS, można zgłosić żądanie dotyczące rozszerzenia (Request for Enhancement-RFE). Produkt IBM może następnie doradzać, czy implementacja jest możliwa w implementacji produktu IBM MQ classes for JMS , czy też istnieje najlepsza praktyka, którą można zastosować. W przypadku dodatkowych funkcji przesyłania komunikatów, ponieważ produkt IBM jest kontrybutorem w otwartym standardzie, funkcje te mogą być zgłaszane jako część procesu JCP.

Zadania pokrewne

Śledzenie aplikacji IBM MQ classes for JMS

Rozwiązywanie problemów z Java i JMS

Informacje pokrewne

Proces podmisji IBM RFE

Proces przeglądu specyfikacji Java JMS

Korzystanie z interfejsów Java produktu WebSphere MQ w środowiskach J2EE/JEE

Uzyskiwanie klas produktu WebSphere MQ dla usługi JMS

Wysyłanie komunikatów PCF za pomocą usługi JMS

Wymagania wstępne dla produktu IBM MQ classes for JMS

W tej sekcji przedstawiono informacje, które należy znać przed użyciem produktu IBM MQ classes for JMS. Aby tworzyć i uruchamiać aplikacje produktu IBM MQ classes for JMS , należy spełnić wymagania wstępne niektórych komponentów oprogramowania.

Więcej informacji na temat wymagań wstępnych dla produktu IBM MQ classes for JMSzawiera sekcja Wymagania systemowe produktu IBM MQ.

Aby tworzyć aplikacje produktu IBM MQ classes for JMS , potrzebny jest pakiet Java SE Software Development Kit (SDK). Szczegółowe informacje na temat pakietów JDK obsługiwanych przez system operacyjny znajdują się w sekcji Wymagania systemowe produktu IBM MQ.

Aby uruchomić aplikacje produktu IBM MQ classes for JMS , potrzebne są następujące komponenty oprogramowania:

- Menedżer kolejek produktu IBM MQ .
- Java runtime environment (JRE) dla każdego systemu, w którym uruchamiane są aplikacje.
- IBM i Dla IBM i, Qshell, który jest opcją 30 systemu operacyjnego.
- z/OS W przypadku produktu z/OS, UNIX and Linux System Services (USS).

Dostawca JSSE IBM zawiera dostawcę kryptograficznego zgodnego ze standardem FIPS, dlatego można go programowo skonfigurować na potrzeby zgodności ze standardem FIPS 140-2, które są gotowe do natychmiastowego użycia. Z tego powodu zgodność ze standardem FIPS 140-2 może być obsługiwana bezpośrednio z produktów IBM MQ classes for Java i IBM MQ classes for JMS.

Dostawca JSSE Oraclemoże mieć skonfigurowanego dostawcę kryptograficznego zgodnego z FIPS, ale nie jest on gotowy do natychmiastowego użycia i nie jest dostępny dla konfiguracji programowej. Dlatego w tym przypadku produkty IBM MQ classes for Java i IBM MQ classes for JMS nie mogą bezpośrednio włączać zgodności ze standardem FIPS 140-2. Może być możliwe ręczne włączenie takiej zgodności, ale produkt IBM nie może obecnie udostępniać wskazówek na ten temat.

Adresy Internet Protocol wersja 6 (IPv6) można używać w aplikacjach IBM MQ classes for JMS , jeśli adresy IPv6 są obsługiwane przez wirtualną maszynę Java (JVM) i implementację protokołu TCP/IP w systemie operacyjnym. Narzędzie administracyjne IBM MQ JMS (patrz sekcja Konfigurowanie obiektów produktu JMS przy użyciu narzędzia administracyjnego). akceptuje również adresy IPv6 .

Narzędzie administracyjne IBM MQ JMS i produkt IBM MQ Explorer używają Java Naming Directory Interface (JNDI) do uzyskania dostępu do usługi katalogowej, w której przechowywane są administrowane obiekty. Aplikacje produktu IBM MQ classes for JMS mogą również używać produktu JNDI do pobierania administrowanych obiektów z usługi katalogowej. Dostawca usług to kod, który zapewnia dostęp do usługi katalogowej przez odwzorowanie wywołań JNDI do usługi katalogowej. Dostawca usług systemu plików w plikach `fscontext.jar` i `providerutil.jar` jest dostarczany razem z produktem IBM MQ classes for JMS. Dostawca usług systemu plików zapewnia dostęp do usługi katalogowej w oparciu o lokalny system plików.

Jeśli planowane jest korzystanie z usługi katalogowej opartej 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 można znaleźć w dokumentacji dostarczonej wraz z serwerem.

Instalowanie i konfigurowanie produktu IBM MQ classes for JMS

W tej sekcji opisano katalogi i pliki, które są tworzone podczas instalowania produktu IBM MQ classes for JMS , a także wyjaśniono, jak skonfigurować produkt IBM MQ classes for JMS po instalacji.

Pojęcia pokrewne

“Korzystanie z adaptera zasobów IBM MQ” na stronie 442
Adapter zasobów umożliwia aplikacjom działającym na serwerze aplikacji uzyskiwanie dostępu do zasobów produktu IBM MQ . Obsługuje komunikację przychodzącą i wychodzącą.

Co jest zainstalowane w systemie IBM MQ classes for JMS

Podczas instalowania produktu IBM MQ classes for JMStworzona jest pewna liczba plików i katalogów. W systemie Windowsniektóre konfiguracje są wykonywane podczas instalacji, automatycznie ustawiając zmienne środowiskowe. Na innych platformach i w niektórych środowiskach Windows należy ustawić zmienne środowiskowe, aby możliwe było uruchamianie aplikacji produktu IBM MQ classes for JMS .

W przypadku większości systemów operacyjnych produkt IBM MQ classes for JMS jest instalowany jako opcjonalny komponent podczas instalowania produktu IBM MQ.

Więcej informacji na temat instalowania produktu IBM MQzawiera sekcja:

- Multi Instalowanie oprogramowania IBM MQ
- z/OS Instalowanie oprogramowania IBM MQ for z/OS

Ważne: Oprócz przemieszczalnych plików JAR opisanych w “IBM MQ classes for JMS relokacyjne pliki JAR” na stronie 89, kopiowanie plików JAR IBM MQ classes for JMS lub rodzimych bibliotek do innych komputerów lub do innego położenia na komputerze, na którym zainstalowano produkt IBM MQ classes for JMS , nie jest obsługiwane.

Katalogi instalacyjne

Tabela 5 na stronie 88 pokazuje miejsce, w którym instalowane są pliki IBM MQ classes for JMS na każdej platformie.

Tabela 5. IBM MQ classes for JMS Katalogi instalacyjne	
Platforma	**Katalog**
Linux UNIX UNIX and Linux	*MQ_INSTALLATION_PATH*/java
Windows Windows	*MQ_INSTALLATION_PATH*\java
IBM i IBM i	/QIBM/ProdData/mqm/java
z/OS z/OS	*MQ_INSTALLATION_PATH*/mqm/V9R1M0/java
	MQ_INSTALLATION_PATH/opt/mqm/java

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Katalog instalacyjny zawiera następującą treść:

- Pliki JAR IBM MQ classes for JMS , w tym relokacyjne pliki JAR, które znajdują się w katalogu *MQ_INSTALLATION_PATH*\java\lib .
- Rodzime biblioteki produktu IBM MQ , które są używane przez aplikacje, które korzystają z interfejsu rodzimego 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 rodzimych bibliotek produktu IBM MQ zawiera sekcja “Konfigurowanie bibliotek rodzimego interfejsu produktu Java (JNI)” na stronie 93.
- Dodatkowe skrypty, które są opisane w sekcji “Skrypty dostarczane z produktem IBM MQ classes for JMS” na stronie 120. Te skrypty znajdują się w katalogu *MQ_INSTALLATION_PATH*\java\bin .
- Specyfikacje interfejsu API produktu IBM MQ classes for JMS . Narzędzie Javadoc zostało użyte do wygenerowania stron HTML, które zawierają specyfikacje interfejsu API.

  The HTML pages are in the *MQ_INSTALLATION_PATH*\java\doc\WMQJMSClasses directory:

  – ULW W systemie UNIX, Linux, and Windowsten podkatalog zawiera poszczególne 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 są opisane w sekcji “Obsługa środowiska OSGi” na stronie 121.
- Adapter zasobów produktu IBM MQ , który może zostać wdrożony na dowolnym serwerze aplikacji zgodnym z produktem Java Platform, Enterprise Edition 7 (Java EE 7).

  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 442 .
- Windows W systemie Windowssymbole, które mogą być używane do debugowania, są instalowane w katalogu *MQ_INSTALLATION_PATH*\java\lib\symbols .

Katalog instalacyjny zawiera również niektóre pliki należące do innych komponentów produktu IBM MQ .

Aplikacje przykładowe

Niektóre aplikacje przykładowe są dostarczane wraz z produktem IBM MQ classes for JMS. Tabela 6 na stronie 89 pokazuje, gdzie przykładowe aplikacje są zainstalowane na każdej platformie.

Tabela 6. Katalogi przykładów

Platforma	Katalog
AIX UNIX 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*/mqm/V9R1M0/java/samples/jms
	MQ_INSTALLATION_PATH/opt/mqm/samp/jms

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Po zakończeniu instalacji może być konieczne wykonanie niektórych zadań konfiguracyjnych w celu skompilowania i uruchomienia aplikacji.

“Ustawianie zmiennych środowiskowych dla IBM MQ classes for JMS” na stronie 91 zawiera opis ścieżki klasy, która jest wymagana do uruchamiania prostych aplikacji produktu IBM MQ classes for JMS . W tym temacie opisano także dodatkowe pliki JAR, do których należy się odwoływać w szczególnych okolicznościach, oraz zmienne środowiskowe, które należy ustawić w celu uruchomienia skryptów dostarczanych z produktem IBM MQ classes for JMS.

Aby sterować właściwościami, takimi jak śledzenie i rejestrowanie aplikacji, należy podać plik właściwości konfiguracyjnych. Plik właściwości konfiguracyjnych produktu IBM MQ classes for JMS jest opisany w sekcji “Plik konfiguracyjny IBM MQ classes for JMS” na stronie 95.

Pojęcia pokrewne

Problemy podczas wdrażania adaptera zasobów

Zadania pokrewne

“Korzystanie z przykładowych aplikacji produktu IBM MQ classes for JMS” na stronie 117
Przykładowe aplikacje produktu IBM MQ classes for JMS zawierają przegląd wspólnych funkcji interfejsu API produktu JMS . Można ich użyć do zweryfikowania konfiguracji serwera instalacji i przesyłania komunikatów oraz do pomocy przy tworzeniu własnych aplikacji.

IBM MQ classes for JMS relokacyjne pliki JAR

Możliwe do ponownego przydzielenia pliki JAR można przenieść do systemów, które muszą uruchamiać program IBM MQ classes for JMS.

Ważne:

- Oprócz przemieszczalnych plików JAR opisanych w sekcji Pliki JAR do przemieszczenia, kopiowanie plików JAR IBM MQ classes for JMS lub rodzimych bibliotek do innych komputerów lub do innego miejsca na komputerze, na którym zainstalowano produkt IBM MQ classes for JMS , nie jest obsługiwane.
- Nie należy uwzględniać relokacyjnych plików JAR w aplikacjach wdrożonych na serwerach aplikacji Java EE , takich jak WebSphere Application Server lub WebSphere Liberty. W tych środowiskach adapter zasobów produktu IBM MQ powinien zostać wdrożony i używany zamiast niego. Należy zauważyć, że produkt WebSphere Application Server osadza adapter zasobów produktu IBM MQ , dlatego nie ma potrzeby ręcznego wdrażania tego adaptera w tym środowisku.
- Aby uniknąć konfliktów programu ładującego klasy, nie zaleca się pakowaniu przemieszczalnych plików JAR w obrębie wielu aplikacji w tym samym środowisku wykonawczym produktu Java .

W tym scenariuszu należy utworzyć pliki JAR relokacyjne produktu IBM MQ dostępne w ścieżce klasy środowiska wykonawczego produktu Java .

- W przypadku tworzenia pakunków przemieszczalnych plików JAR w aplikacjach, należy uwzględnić wszystkie wstępnie wymagane pliki JAR zgodnie z opisem w sekcji Pliki JAR z produktem Relocatable JAR. Należy również upewnić się, że istnieją odpowiednie procedury aktualizacji dołączonych plików JAR w ramach obsługi aplikacji, aby upewnić się, że IBM MQ classes for JMS pozostają aktualne, a znane problemy są ponownie mediowane.

Przemieszczalne pliki JAR

W obrębie przedsiębiorstwa można przenosić następujące pliki do systemów, które muszą uruchamiać program IBM MQ classes for JMS:

- `com.ibm.mq.allclient.jar`
- `com.ibm.mq.traceControl.jar`
- `jms.jar`
- `fscontext.jar`
- `providerutil.jar`
- `bcpkix-jdk15on.jar`
- `bcprov-jdk15on.jar`
- V 9.1.0.9 `bcutil-jdk15on.jar`
- V 9.1.2 `org.json.jar`

Plik `jms.jar` jest plikiem interfejsu dla `javax.jms.jar` i zawiera klasy inne niżIBM JMS .

Pliki `fscontext.jar` i `providerutil.jar` są wymagane, jeśli aplikacja wykonuje wyszukiwanie JNDI przy użyciu kontekstu systemu plików.

Wymagane są pliki JAR obsługi zabezpieczeń Zamku Bouncy oraz pliki JAR obsługi CMS. Więcej informacji na ten temat zawiera sekcja Wsparcie dla środowisk JRE innych niżIBM z AMS. Wymagane są następujące pliki JAR:

- `bcpkix-jdk15on.jar`
- `bcprov-jdk15on.jar`
- V 9.1.0.9 `bcutil-jdk15on.jar`

V 9.1.2 Plik `org.json.jar` jest wymagany, jeśli aplikacja IBM MQ classes for JMS korzysta z tabeli definicji kanału klienta w formacie JSON.

Plik `com.ibm.mq.allclient.jar` zawiera IBM MQ classes for JMS, IBM MQ classes for Javaoraz klasy PCF i Headers. Jeśli ten plik zostanie przeniesiony do nowego położenia, należy się upewnić, że nowe położenie zostało zachowane przy użyciu nowych pakietów poprawek produktu IBM MQ , które zostały zachowane. Należy również upewnić się, że użycie tego pliku jest znane z działu wsparcia produktu IBM , jeśli jest otrzymana poprawka tymczasowa.

Aby określić wersję pliku `com.ibm.mq.allclient.jar`, należy użyć następującej komendy:

```
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:    9.1.0.0
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:    9.1.0.0
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:    9.1.0.0
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:    9.1.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar
```

Plik `com.ibm.mq.traceControl.jar` jest używany do dynamicznego sterowania śledzeniem dla aplikacji IBM MQ classes for JMS . Więcej informacji na ten temat zawiera sekcja Kontrolowanie śledzenia w działającym procesie przy użyciu klas produktu IBM MQ dla języka Java i klas produktu IBM MQ dla usługi JMS.

Ustawianie zmiennych środowiskowych dla IBM MQ classes for JMS
Przed skompilowaniem i uruchomieniem aplikacji IBM MQ classes for JMS ustawienie zmiennej środowiskowej CLASSPATH musi zawierać plik archiwum IBM MQ classes for JMS Java (JAR). 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, należy ustawić inne zmienne środowiskowe.

O tym zadaniu

Ważne: Ustawienie opcji Java *-Xbootclasspath*na wartość IBM MQ classes for JMSnie jest obsługiwane.

Aby skompilować i uruchomić aplikacje IBM MQ classes for JMS , należy użyć ustawienia CLASSPATH dla używanej platformy, jak to pokazano na rysunku (Tabela 7 na stronie 91). To ustawienie obejmuje katalog przykładów, który umożliwia kompilowanie i uruchamianie przykładowych aplikacji IBM MQ classes for JMS . Zamiast używać zmiennej środowiskowej można również określić ścieżkę klasy w komendzie **java** .

Tabela 7. Ustawienie CLASSPATH podczas kompilowania i uruchamiania aplikacji IBM MQ classes for JMS , w tym aplikacji przykładowych

Platforma	Ustawienie CLASSPATH
AIX	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mqjms.jar: , *MQ_INSTALLATION_PATH*/samp/jms/samples:
Linux Solaris	CLASSPATH= *MQ_INSTALLATION_PATH*/java/lib/com.ibm.mqjms.jar: , *MQ_INSTALLATION_PATH*/samp/jms/samples:
IBM i	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mqjms.jar: /QIBM/ProdData/mqm/java/samples/jms/samples:
Windows	CLASSPATH= *MQ_INSTALLATION_PATH*\java\lib\com.ibm.mqjms.jar; *MQ_INSTALLATION_PATH*\tools\jms\samples;

Tabela 7. Ustawienie CLASSPATH podczas kompilowania i uruchamiania aplikacji IBM MQ classes for JMS , w tym aplikacji przykładowych (kontynuacja)	
Platforma	**Ustawienie CLASSPATH**
z/OS z/OS	LTS CLASSPATH= *MQ_INSTALLATION_PATH*/mqm/V9R0M0/java/lib/com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/mqm/V9R0M0/java/samples/jms/samples: CD CLASSPATH= *MQ_INSTALLATION_PATH*/mqm/V9R1M5/java/lib/com.ibm.mqjms.jar: *MQ_INSTALLATION_PATH*/mqm/V9R1M5/java/samples/jms/samples:

MQ_INSTALLATION_PATH reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

Manifest pliku JAR `com.ibm.mqjms.jar` zawiera odwołania do większości innych plików JAR wymaganych przez aplikacje IBM MQ classes for JMS , 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 dostarczone z programem IBM MQ classes for JMS używają następujących zmiennych środowiskowych:

MQ_JAVA_DATA_PATH,
Ta zmienna środowiskowa określa katalog dla danych wyjściowych dziennika i śledzenia.

ŚCIEŻKA_INSTALACJI_PRODUKTU_MQJAVA
Ta zmienna środowiskowa określa katalog, w którym zainstalowano produkt IBM MQ classes for JMS .

MQ_JAVA_LIB_PATH
Ta zmienna środowiskowa określa katalog, w którym są przechowywane biblioteki produktu IBM MQ classes for JMS (patrz Tabela 8 na stronie 94).

Procedura

- Windows
  W systemie Windowspo zainstalowaniu produktu IBM MQnależy uruchomić komendę **setmqenv**.
  Jeśli ta komenda nie zostanie uruchomiona jako pierwsza, podczas uruchamiania komendy **dspmqver** może zostać wyświetlony następujący komunikat o błędzie:

V 9.1.0 AMQ8351: IBM MQ
lub składnik IBM MQ JRE nie został zainstalowany.

Uwaga: V 9.1.0 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).

- Na dowolnej innej platformie należy samodzielnie ustawić zmienne środowiskowe:
  - Linux UNIX Aby ustawić zmienne środowiskowe, jeśli używana jest 32-bitowa maszyna JVM w systemach UNIX, lub Linux , można użyć skryptu setjmsenv.
  - Linux UNIX Aby ustawić zmienne środowiskowe w przypadku używania 64-bitowej maszyny JVM w systemie UNIX lub Linux , można użyć skryptu setjmsenv64. 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 .

  Można użyć skryptu setjmsenv lub setjmsenv64 na wiele sposobów: można go 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 . Jeśli ta opcja zostanie wybrana, należy uruchomić skrypt w każdym oknie powłoki, które zostanie uruchomione, podczas procesu weryfikacji JMS , wpisując komendę `. ./setjmsenv` lub `. ./setjmsenv64` .

  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 językiem Java i usługą JMS .

Konfigurowanie bibliotek rodzimego interfejsu produktu Java (JNI)
Aplikacje produktu 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 uruchamiane w środowisku, które umożliwia dostęp do bibliotek JNI (Native Interface) produktu Java .

Zanim rozpoczniesz

Więcej informacji na temat korzystania ze 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, tak aby produkt Java virtual machine (JVM) mógł załadować bibliotekę mqjbnd przed uruchomieniem aplikacji IBM MQ classes for JMS .

Produkt IBM MQ udostępnia dwie biblioteki interfejsu 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 serwerem IBM MQ classes for JMS i menedżerem kolejek. Biblioteka mqjbnd zainstalowana razem z produktem IBM MQ 9.0 może być używana do nawiązywania połączenia z dowolnym menedżerem kolejek produktu IBM MQ 9.0 (lub wcześniejszymi).

mqjexitstub02
Biblioteka mqjexitstub02 jest ładowana przez składnik IBM MQ classes for JMS , gdy aplikacja łączy się z menedżerem kolejek przy użyciu transportu klienta i korzysta z 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 jest przedstawione w Tabeli 1.

Tabela 8. Położenie bibliotek produktu IBM MQ classes for JMS dla każdej platformy

Platforma	**Katalog zawierający biblioteki produktu IBM MQ classes for JMS**
AIX AIX Linux Linux (platformy POWER, x86-64 i zSeries s390x) Solaris Solaris (platformyx86-64 i SPARC)	*MQ_INSTALLATION_PATH*/java/lib (biblioteki 32-bitowe) *MQ_INSTALLATION_PATH*/java/lib64 (biblioteki 64-bitowe)
Windows Windows	*MQ_INSTALLATION_PATH*\java\lib (biblioteki 32-bitowe) *MQ_INSTALLATION_PATH*\java\lib64 (biblioteki 64-bitowe)
z/OS z/OS	*MQ_INSTALLATION_PATH*/mqm/V9R1M0/java/lib (biblioteki 31-bitowe i 64-bitowe)

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Uwaga: z/OS W systemie z/OSmożna korzystać z 31-bitowego lub 64-bitowego produktu Java virtual machine (JVM). Nie trzeba określać, które biblioteki JNI mają być używane. IBM MQ classes for JMS może określić, które biblioteki JNI mają zostać załadowane.

Procedura

1. Skonfiguruj właściwość **java.library.path** maszyny JVM, która może być wykonana na dwa sposoby:
   - Określając argument maszyny JVM w sposób przedstawiony 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 położenia należy określić:

     ```
     -Djava.library.path=/opt/mqm/java/lib64
     ```

   - Skonfigurowanie środowiska powłoki w taki sposób, aby maszyna JVM skonfigurowała własny serwer `java.library.path`. Ta ścieżka różni się w zależności od platformy i położenia, w którym zainstalowano produkt IBM MQ. Na przykład w przypadku 64-bitowej maszyny JVM i domyślnej lokalizacji instalacji produktu IBM MQ można użyć następujących ustawień:

     AIX
     ```
     export LIBPATH=/usr/mqm/java/lib64:$LIBPATH
     ```

     Solaris 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 używany w sytuacji, gdy środowisko nie zostało poprawnie skonfigurowane:

```
Spowodowane przez: com.ibm.mq.jmqi.local.LocalMQ$4: CC=2;RC=2495;
AMQ8598: Nie powiodła się próba załadowania rodzimej biblioteki JNI produktu WebSphere MQ :
mqjbnd.
    w pliku com.ibm.mq.jmqi.local.LocalMQ.loadLib(LocalMQ.java:1268)
    w pliku com.ibm.mq.jmqi.local.LocalMQ$1.run(LocalMQ.java:309)
    na poziomie java.security.AccessController.doPrivileged(AccessController.java:400)
    w pliku com.ibm.mq.jmqi.local.LocalMQ.initialise_inner(LocalMQ.java:259)
    w pliku com.ibm.mq.jmqi.local.LocalMQ.initialise(LocalMQ.java:221)
    w pliku com.ibm.mq.jmqi.local.LocalMQ. < init> (LocalMQ.java:1350)
    w pliku com.ibm.mq.jmqi.local.LocalServer. < init> (LocalServer.java:230)
    at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
    w pliku
sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:86)

    w pliku
sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.jav
a:58)
    w klasie java.lang.reflect.Constructor.newInstance(Constructor.java:542)
    w metodzie com.ibm.mq.jmqi.JmqiEnvironment.getInstance(JmqiEnvironment.java:706)
    w elemencie com.ibm.mq.jmqi.JmqiEnvironment.getMQI(JmqiEnvironment.java:640)
    w pliku
com.ibm.msg.client.wmq.factories.WMQConnectionFactory.createV7ProviderConnection(WMQConnectionF
actory.java:8437)
    ... 7 więcej
Spowodowany przez: java.lang.UnsatisfiedLinkError: mqjbnd (Nie znaleziono w pliku
java.library.path)
    w klasie java.lang.ClassLoader.loadLibraryWithPath(ClassLoader.java:1235)
    w klasie java.lang.ClassLoader.loadLibraryWithClassLoader(ClassLoader.java:1205)
    w java.lang.System.loadLibrary(System.java:534)
    w pliku com.ibm.mq.jmqi.local.LocalMQ.loadLib(LocalMQ.java:1240)
    ... 20 więcej
```

2. Po ustawieniu środowiska 32-lub 64-bitowego uruchom aplikację IBM MQ classes for JMS , używając komendy:

   ```
   java application-name
   ```

   gdzie *nazwa_aplikacji* to nazwa aplikacji IBM MQ classes for JMS , która ma zostać uruchomiona.

   Wyjątek zawierający kod przyczyny IBM MQ 2495 (MQRC_MODULE_NOT_FOUND) jest zgłaszany przez IBM MQ classes for JMS , jeśli:

   - Aplikacja IBM MQ classes for JMS jest uruchamiana w 32-bitowej wersji produktu Java runtime environment, a dla partycji IBM MQ classes for JMSzostało skonfigurowane środowisko 64-bitowe, ponieważ 32-bitowa wersja Java runtime environment nie może załadować 64-bitowej biblioteki rodzimej produktu Java .
   - Aplikacja IBM MQ classes for JMS jest uruchamiana w 64-bitowym systemie Java runtime environment, a środowisko 32-bitowe zostało skonfigurowane dla partycji IBM MQ classes for JMS, ponieważ 64-bitowa wersja Java runtime environment nie może załadować 32-bitowej biblioteki rodzimej produktu Java .

Plik konfiguracyjny IBM MQ classes for JMS
Plik konfiguracyjny IBM MQ classes for JMS określa właściwości, które są używane do konfigurowania produktu IBM MQ classes for JMS.

Uwaga: Właściwości zdefiniowane w pliku konfiguracyjnym mogą być również ustawione jako właściwości systemowe maszyny JVM. Jeśli właściwość jest ustawiona zarówno w pliku konfiguracyjnym, jak i jako właściwość systemowa, to właściwość systemowa ma pierwszeństwo. Dlatego jeśli jest to wymagane, można przesłonić dowolną właściwość w pliku konfiguracyjnym, określając ją jako właściwość systemową w komendzie **java** .

Format pliku konfiguracyjnego IBM MQ classes for JMS jest taki sam jak standardowy plik właściwości Java . Przykładowy plik konfiguracyjny o nazwie `jms.config` jest dostarczany w podkatalogu `bin` w katalogu instalacyjnym IBM MQ classes for JMS . Ten plik dokumentuje wszystkie obsługiwane właściwości i ich wartości domyślne.

Użytkownik może wybrać nazwę i położenie pliku konfiguracyjnego produktu IBM MQ classes for JMS . Po uruchomieniu aplikacji należy użyć komendy **java** z następującym formatem:

```
java -Dcom.ibm.msg.client.config.location= config_file_url application_name
```

W komendzie *config_file_url* jest jednostajnym lokalizatorem zasobów (URL), który określa nazwę i położenie pliku konfiguracyjnego IBM MQ classes for JMS . Obsługiwane są adresy URL następujących typów: 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 jako plik `D:\mydir\mjms.config` w lokalnym systemie Windows .

Po uruchomieniu aplikacji program IBM MQ classes for JMS odczytuje treść pliku konfiguracyjnego i zapisuje określone właściwości w wewnętrznej składnicy właściwości. Jeśli komenda **java** nie identyfikuje pliku konfiguracyjnego lub jeśli nie można znaleźć pliku konfiguracyjnego, program IBM MQ classes for JMS użyje wartości domyślnych dla wszystkich właściwości.

Plik konfiguracyjny IBM MQ classes for JMS może być używany z dowolnym z obsługiwanych transportów między aplikacją a menedżerem kolejek lub brokerem.

Nadpisywanie właściwości określonych w pliku konfiguracyjnym IBM MQ MQI client

Plik konfiguracyjny produktu IBM MQ MQI client może również określać właściwości, które są używane do konfigurowania produktu IBM MQ classes for JMS. Jednak właściwości określone w pliku konfiguracyjnym produktu IBM MQ MQI client mają zastosowanie tylko wtedy, gdy aplikacja łączy się z menedżerem kolejek w trybie klienta.

Jeśli jest to wymagane, można przesłonić dowolny atrybut w pliku konfiguracyjnym IBM MQ MQI client , określając go jako właściwość w pliku konfiguracyjnym IBM MQ classes for JMS . Aby przesłonić atrybut w pliku konfiguracyjnym produktu IBM MQ MQI client , należy użyć wpisu o następującym formacie w pliku konfiguracyjnym produktu IBM MQ classes for JMS :

```
com.ibm.mq.cfg. stanza. propName = propValue
```

Zmienne w pozycji mają następujące znaczenia:

sekcja
Nazwa sekcji w pliku konfiguracyjnym IBM MQ MQI client , który zawiera atrybut

propName
Nazwa atrybutu określona w pliku konfiguracyjnym IBM MQ MQI client .

propValue
Wartość właściwości, która nadpisuje wartość atrybutu określonego w pliku konfiguracyjnym IBM MQ MQI client .

Alternatywnie można przesłonić atrybut w pliku konfiguracyjnym IBM MQ MQI client , określając właściwość jako właściwość systemową w komendzie **java** . Aby określić właściwość jako właściwość systemową, należy użyć poprzedniego formatu.

Tylko następujące atrybuty w pliku konfiguracyjnym IBM MQ MQI client mają znaczenie dla produktu IBM MQ classes for JMS. Jeśli zostaną określone lub nadpisane inne atrybuty, nie będzie to miało żadnego wpływu. W szczególności należy pamiętać, że produkty `ChannelDefinitionFile` i `ChannelDefinitionDirectory` w sekcji CHANNELS w pliku konfiguracyjnym klienta nie są używane. Szczegółowe informacje na temat korzystania z tabeli definicji kanału klienta z IBM MQ classes for JMSmożna znaleźć w sekcji “Korzystanie z tabeli definicji kanału klienta przy użyciu produktu IBM MQ classes for JMS” na stronie 280 .

Tabela 9. Która sekcja pliku konfiguracyjnego klienta zawiera atrybut, który atrybut	
sekcja	**Atrybut**
Sekcja CHANNELS pliku konfiguracyjnego klienta	Put1DefaultAlwaysSync
Sekcja CHANNELS pliku konfiguracyjnego klienta	DefRecon
Sekcja CHANNELS pliku konfiguracyjnego klienta	ReconDelay
Sekcja CHANNELS pliku konfiguracyjnego klienta	PasswordProtection
ClientExitSekcja ścieżki pliku konfiguracyjnego klienta	ExitsDefaultPath
ClientExitSekcja ścieżki pliku konfiguracyjnego klienta	ExitsDefaultPath64
ClientExitSekcja ścieżki pliku konfiguracyjnego klienta	JavaExitsClasspath
Sekcja JMQI pliku konfiguracyjnego klienta	useMQCSPauthentication
Sekcja MessageBuffer pliku konfiguracyjnego klienta	MaximumSize
Sekcja MessageBuffer pliku konfiguracyjnego klienta	PurgeTime
Sekcja MessageBuffer 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

Szczegółowe informacje na temat konfigurowania produktu IBM MQ MQI client zawiera sekcja Konfigurowanie klienta przy użyciu pliku konfiguracyjnego .

Sekcja Java Standard Environment Trace
Aby skonfigurować narzędzie śledzenia produktu IBM MQ classes for JMS , należy użyć sekcji Standard Trace Settings (Ustawienia śledzenia środowiska) produktu Java .

com.ibm.msg.client.commonservices.trace.outputName = *traceOutputNazwa*
traceOutputName to katalog i nazwa pliku, do którego wysyłane są 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 przypadku produktu IBM MQ classes for JMS dla systemu IBM MQ 9.0.0 Fix Pack 1 lub wcześniejszej informacje śledzenia są zapisywane w pliku o nazwie `mqjms_%PID%.trc`.
- W produkcie IBM MQ 9.0.0 Fix Pack 2, jeśli aplikacja załadowała IBM MQ classes for JMS z pliku JAR `com.ibm.mqjms.jar`, dane śledzenia są zapisywane w pliku o nazwie `mqjava_%PID%.trc`.
- W produkcie IBM MQ 9.0.0 Fix Pack 2, jeśli aplikacja załadowała 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`.
- V 9.1.5 V 9.1.0.5 W przypadku IBM MQ 9.1.5 i IBM MQ 9.1.0 Fix Pack 5, jeśli aplikacja załadowała 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`.

- V 9.1.5 V 9.1.0.5 W przypadku IBM MQ 9.1.5 i IBM MQ 9.1.0 Fix Pack 5, jeśli aplikacja załadowała 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`.

gdzie *%PID%* jest identyfikatorem procesu, który jest śledzony, a *%u* jest unikalną liczbą w celu odróżnienia plików między wątkami uruchamiających śledzenie w różnych programach ładujących klasy Java .

Jeśli identyfikator procesu jest niedostępny, generowany jest numer losowy, który jest poprzedzony literą `f`. Aby dołączyć identyfikator procesu do określonej nazwy pliku, należy użyć łańcucha `%PID%`.

Jeśli zostanie podany katalog alternatywny, musi on istnieć, a użytkownik musi mieć uprawnienia do zapisu dla tego katalogu. Jeśli użytkownik nie ma uprawnień do zapisu, dane wyjściowe śledzenia są zapisywane w produkcie `System.err`.

com.ibm.msg.client.commonservices.trace.include = *includeList*
includeList to lista śledzonych pakietów i klas albo wartości specjalne `ALL` lub `NONE`.

Oddziel nazwy pakietów lub klas średnikami, `;`. *includeList* przyjmuje wartości domyślne `ALL`i śledzi wszystkie pakiety i klasy w produkcie IBM MQ classes for JMS.

Uwaga: Możliwe jest dołączenie pakietu, a następnie wykluczenie podpakietów tego pakietu. For example, if you include package `a.b` and exclude package `a.b.x`, the trace includes everything in `a.b.y` and `a.b.z`, but not `a.b.x` or `a.b.x.1`.

com.ibm.msg.client.commonservices.trace.exclude = *excludeList*
excludeList to lista pakietów i klas, które nie są śledzone, lub wartości specjalne `ALL` lub `NONE`.

Oddziel nazwy pakietów lub klas średnikami, `;`. Wartością domyślną *excludeList* jest `NONE`, dlatego nie jest wykluczany żaden pakiet i klasy w produkcie IBM MQ classes for JMS .

Uwaga: Użytkownik może wykluczyć pakiet, ale następnie uwzględnić podpakiety tego pakietu. Jeśli na przykład pakiet `a.b` zostanie wykluczony i dołączany jest pakiet `a.b.x`, dane śledzenia będą zawierały wszystkie elementy w produkcie `a.b.x` i `a.b.x.1`, ale nie będą zawierać wartości `a.b.y` ani `a.b.z`.

Uwzględniane są wszystkie pakiety lub klasy, które są określone na tym samym poziomie, co zarówno włączone, jak i wykluczone.

com.ibm.msg.client.commonservices.trace.maxBytes = *maxArrayBajty*
maxArrayBytes to maksymalna liczba bajtów, które są śledzone z dowolnej tablicy bajtów.

Jeśli parametr *maxArrayBytes* jest ustawiony na dodatnią liczbę całkowitą, ogranicza liczbę bajtów w tablicy bajtów, które są zapisywane w pliku śledzenia. Powoduje obcięcie tablicy bajtów po wypisaniu *maxArrayBytes* . Ustawienie *maxArrayBytes* redukuje wielkość wynikowego pliku śledzenia i zmniejsza wpływ śledzenia na wydajność aplikacji.

Wartość `0` dla tej właściwości oznacza, że żadna z treści żadnej tablicy bajtów nie jest wysyłana do pliku śledzenia.

Wartością domyślną jest `-1`, co powoduje usunięcie dowolnego 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 zapisanych w pliku wyjściowym śledzenia.

Produkt *maxTraceBytes* współpracuje z produktem *traceCycles*. Jeśli liczba zapisanych bajtów śledzenia znajduje się w pobliżu limitu, plik jest zamykany, a nowy plik wyjściowy śledzenia jest uruchamiany.

Wartość `0` oznacza, że plik wyjściowy śledzenia ma zerową długość. Wartością domyślną jest `-1`, co oznacza, że ilość danych, które mają zostać zapisane w pliku wyjściowym śledzenia, jest nieograniczona.

com.ibm.msg.client.commonservices.trace.count = *traceCycles*
traceCycles to liczba plików wyjściowych śledzenia, przez które mają zostać przełączone dane wyjściowe.

Jeśli bieżący plik wyjściowy śledzenia osiągnie limit określony przez parametr *maxTraceBytes*, plik zostanie zamknięty. Dalsze dane wyjściowe śledzenia są zapisywane w kolejnym pliku wyjściowym śledzenia w kolejności. Każdy plik wyjściowy śledzenia jest wyróżniany przyrostkiem liczbowym dodanym do nazwy pliku. Bieżący lub najnowszy plik wyjściowy śledzenia to `mqjms.trc.0`, następny najnowszy 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 parametr *traceCycles* ma wartość 1, wówczas gdy bieżący plik wyjściowy śledzenia osiągnie swoją maksymalną wielkość, plik jest zamykany i usuwany. Zostanie uruchomiony nowy plik wyjściowy śledzenia o tej samej nazwie. Oznacza to, że w danym momencie istnieje tylko jeden plik wyjściowy śledzenia.

com.ibm.msg.client.commonservices.trace.parameter = *traceParameters*
traceParameters określa, czy parametry metody i wartości zwracane są uwzględniane w śledzeniu.

Wartością domyślną *traceParameters* jest TRUE. Jeśli parametr *traceParameters* jest ustawiony na wartość FALSE, śledzone są tylko sygnatury metod.

com.ibm.msg.client.commonservices.trace.startup = *startup*
Jest to faza inicjowania IBM MQ classes for JMS , podczas której przydzielane są zasoby. Główny narzędzie śledzenia jest inicjowany podczas fazy przydzielania zasobów.

Jeśli parametr *startup* jest ustawiony na wartość TRUE, używane jest śledzenie uruchamiania. Informacje śledzenia są generowane natychmiast i obejmują konfigurację wszystkich komponentów, w tym również funkcję śledzenia. Informacje śledzenia uruchamiania mogą być używane do diagnozowania problemów z konfiguracją. Informacje śledzenia uruchamiania są zawsze zapisywane w produkcie `System.err`.

Wartością domyślną *startup* jest FALSE.

startup jest sprawdzane przed ukończonainicjalizacją. Z tego powodu jako właściwość systemową Java należy określić właściwość w wierszu komend. Nie podauj go w pliku konfiguracyjnym IBM MQ classes for JMS .

com.ibm.msg.client.commonservices.trace.compress = *compressedTrace*
Ustaw opcję *compressedTrace* na TRUE , aby skompresować dane wyjściowe śledzenia.

Wartością domyślną parametru *compressedTrace* jest FALSE.

Jeśli parametr *compressedTrace* jest ustawiony na wartość TRUE, dane wyjściowe śledzenia są kompresowane. Domyślna nazwa pliku wyjściowego śledzenia ma rozszerzenie `.trz`. Jeśli kompresja jest ustawiona na wartość FALSE(wartość domyślna), plik ma rozszerzenie `.trc` , aby wskazać, że jest on nieskompresowany. Jeśli jednak nazwa pliku dla danych wyjściowych śledzenia została określona w polu *traceOutputName* , to zamiast tego zostanie użyta nazwa pliku. Do tego pliku nie zostanie zastosowany przyrostek.

Skompresowane dane wyjściowe śledzenia są mniejsze niż nieskompresowane. Ponieważ istnieje mniej operacji we/wy, można je zapisać szybciej niż nieskompresowane śledzenie. Śledzenie skompresowane ma mniejszy wpływ na wydajność programu IBM MQ classes for JMS niż nieskompresowane śledzenie.

Jeśli ustawione są opcje *maxTraceBytes* i *traceCycles* , to w miejsce wielu plików tekstowych tworzone są wiele skompresowanych plików śledzenia.

Jeśli produkt IBM MQ classes for JMS zostanie zakończony w sposób niekontrolowany, skompresowany plik śledzenia może nie być poprawny. Z tego powodu kompresja śledzenia musi być używana tylko wtedy, gdy produkt IBM MQ classes for JMS zostanie zamknięty w kontrolowany sposób. Kompresja śledzenia jest używana tylko wtedy, gdy badane problemy nie powodują nieoczekiwanego zatrzymania maszyny JVM. Nie należy używać kompresji śledzenia podczas diagnozowania problemów, które mogą spowodować zamknięcie systemu `System.Halt()` lub nieprawidłowe, niekontrolowane zakończenie 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, to dowolny punkt śledzenia ze zdefiniowanym poziomem TRACE_EXCEPTION, TRACE_WARNINGlub TRACE_INFO jest zapisywany w śledzeniu. Wszystkie pozostałe punkty śledzenia są wykluczone.

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* jest ustawiony na wartość TRUE, właściwości śledzenia klienta IBM MQ JMS są używane do określenia konfiguracji śledzenia.

Jeśli parametr *standaloneTrace* jest ustawiony na wartość FALSE, a klient IBM MQ JMS jest uruchomiony w kontenerze WebSphere Application Server , używana jest usługa śledzenia produktu WebSphere Application Server . Generowane informacje śledzenia zależą od ustawień śledzenia serwera aplikacji.

Wartością domyślną parametru *standaloneTrace* jest FALSE.

Sekcja rejestrowania
Sekcja Rejestrowanie służy do konfigurowania funkcji dziennika produktu IBM MQ classes for JMS .

W sekcji Rejestrowanie można uwzględnić następujące właściwości:

com.ibm.msg.client.commonservices.log.outputName = ścieżka
Nazwa pliku dziennika używanego przez narzędzie dziennika produktu IBM MQ classes for JMS . Wartością domyślną jest mqjms.log, która jest zapisywana w bieżącym katalogu roboczym dla środowiska wykonawczego Java , w którym działa produkt 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 bezwzględną lub względną nazwą ścieżki lub:

"stderr" lub "System.err"
Reprezentuje standardowy strumień błędów.

"stdout" lub "System.out"
Reprezentuje standardowy strumień wyjściowy.

com.ibm.msg.client.commonservices.log.maxBytes
Maksymalna liczba bajtów, które są rejestrowane z dowolnego wywołania w celu zarejestrowania danych komunikatu.

Dodatnia liczba całkowita
Dane są zapisywane do tej wartości w bajtach na wywołanie dziennika.

0
Żadne dane nie są zapisywane.

-1
Nieograniczone dane są zapisywane (domyślnie).

com.ibm.msg.client.commonservices.log.limit
Maksymalna liczba bajtów zapisanych do dowolnego pliku dziennika (wartość domyślna to 262144).

Dodatnia liczba całkowita
Dane są zapisywane do tej wartości w bajtach na plik dziennika.

0
Żadne dane nie są zapisywane.

-1
Nieograniczone dane są zapisywane.

com.ibm.msg.client.commonservices.log.count
Liczba plików dziennika, przez które mają być przełączane cykle. Ponieważ każdy plik osiągnie śledzenie `com.ibm.msg.client.commonservices.trace.limit` , rozpocznie się on w następnym pliku, a wartością domyślną jest 3.

Dodatnia liczba całkowita
Liczba plików, przez które mają zostać przełączone.

0
Pojedynczy plik.

Sekcja Java SE Specifics
Sekcja Java SE Specifics służy do konfigurowania właściwości, które są używane, gdy produkt IBM MQ classes for JMS jest używany w środowisku Java Standard Edition .

com.ibm.msg.client.commonservices.j2se.produceJavaCore = TRUE|FALSE
Określa, czy plik podstawowy produktu Javajest zapisywany natychmiast po wygenerowaniu pliku FDC przez program IBM MQ classes for JMS . Jeśli wartość ta jest ustawiona na wartość TRUE, plik core produktu Javajest tworzony w katalogu roboczym środowiska wykonawczego produktu Java , w którym działa produkt IBM MQ classes for JMS .

TRUE
Wygeneruj moduł podstawowy produktu Java, pod warunkiem że środowisko wykonawcze produktu Java jest w stanie wykonać tę funkcję.

FALSE
Nie należy generować podstawowego modułu Java. Jest to wartość domyślna.

Sekcja Właściwości produktu IBM MQ
Sekcja Właściwości produktu IBM MQ służy do ustawiania właściwości, które wpływają na sposób interakcji produktu IBM MQ classes for JMS z produktem IBM MQ.

com.ibm.msg.client.wmq.compat.base.internal.MQQueue.smallMsgsBufferReductionThreshold
Gdy aplikacja, która używa IBM MQ classes for JMS łączy się z menedżerem kolejek produktu IBM MQ przy użyciu trybu migracji dostawcy przesyłania komunikatów produktu IBM MQ , produkt IBM MQ classes for JMS używa domyślnej wielkości buforu o wielkości 4 kB, gdy odbiera komunikaty. Jeśli komunikat, który aplikacja próbuje uzyskać, jest większy niż 4 kB, IBM MQ classes for JMS zmienia wielkość buforu tak, aby była na tyle duża, aby pomieścić komunikat. Większa wielkość buforu jest następnie używana, gdy odbierane są kolejne komunikaty.

Ta właściwość określa, kiedy wielkość buforu zostanie zmniejszona z powrotem do 4 kB. Domyślnie po odebraniu dziesięciu kolejnych komunikatów, które są mniejsze od większej wielkości buforu, wielkość buforu zostaje zmniejszona z powrotem do 4 kB. Aby przywrócić wielkość buforu do 4 KB przy każdym odebraniu komunikatu, należy ustawić właściwość na wartość 0.

0

Bufor zawsze resetuje się do wielkości domyślnej.

10

Jest to wartość domyślna. Bufor zostanie zrezygnowany po dziesiątym komunikacie.

com.ibm.msg.client.wmq.receiveConversionCCSID

Jeśli aplikacja używaająca IBM MQ classes for JMS łączy się z menedżerem kolejek produktu IBM MQ przy użyciu trybu normalnego dostawcy przesyłania komunikatów produktu IBM MQ , właściwość `receiveConversionCCSID` może zostać ustawiona w taki sposób, aby przesłonić domyślną wartość identyfikatora CCSID w strukturze MQMD używanej do odbierania komunikatów z menedżera kolejek. Domyślnie MQMD zawiera pole CCSID ustawione na 1208, ale można to zmienić, jeśli na przykład menedżer kolejek nie może przekształcić komunikatów na tę stronę kodową.

Poprawne wartości to dowolny poprawny identyfikator CCSID lub jedna z następujących wartości:

-1

Użyj wartości domyślnej platformy.

1208

Jest to wartość domyślna.

Sekcja specifics klienta w trybie klienta

Sekcja specyfika trybu klienta służy do określania właściwości, które są używane, gdy produkt IBM MQ classes for JMS łączy się z menedżerem kolejek, który korzysta z transportu CLIENT.

com.ibm.mq.polling.RemoteRequestEntry

Określa przedział czasu odpytywania używany przez produkt IBM MQ classes for JMS do sprawdzania zerwanych połączeń, gdy oczekuje na odpowiedź z menedżera kolejek.

Dodatnia liczba całkowita

Liczba milisekund, których należy zaczekać przed sprawdzeniem. Wartością domyślną jest 10000 lub 10 sekund. Minimalna wartość to 3000, a niższe wartości są traktowane w taki sam sposób, jak ta wartość minimalna.

Właściwości używane do konfigurowania zachowania klienta JMS

Za pomocą tych właściwości można skonfigurować zachowanie klienta JMS .

com.ibm.mq.jms.SupportMQExtensions TRUE|FALSE

Specyfikacja JMS 2.0 wprowadza zmiany w sposobie pracy niektórych zachowań. Produkt IBM MQ 8.0 zawiera właściwość `com.ibm.mq.jms.SupportMQExtensions`, która może mieć wartość *TRUE*, aby przywrócić te zmienione zachowania do poprzednich implementacji. Przywracanie zmienionych zachowań może być konieczne dla niektórych aplikacji produktu JMS 2.0 , a także dla niektórych aplikacji, które korzystają z interfejsu API JMS 1.1 , ale działają na serwerze IBM MQ 8.0 IBM MQ classes for JMS.

PRAWDA

Następujące trzy obszary funkcjonalności są odwracane przez ustawienie parametru SupportMQExtensions na wartość TRUE:

Priorytet komunikatu

Do wiadomości można przypisać priorytet, *0* - *9*. Przed JMS 2.0komunikaty mogą również używać wartości *-1*, co wskazuje, 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, które nie mają wartości NULL, były sprawdzane pod kątem unikalności podczas nawiązywania połączenia. Włączenie produktu `SupportMQExtensions`oznacza, że wymaganie to nie jest brane pod uwagę i że identyfikator klienta może zostać ponownie wykorzystany.

NoLocal

Specyfikacja JMS 2.0 wymaga, aby po włączeniu tej stałej konsument nie mógł odbierać komunikatów publikowanych przez ten sam identyfikator klienta. Przed JMS 2.0ten atrybut został ustawiony w subskrybencie, aby uniemożliwić odbieranie komunikatów publikowanych

przez jego własne połączenie. Włączenie produktu SupportMQExtensions powoduje przywrócenie tego działania do jego poprzedniej implementacji.

FALSE
Zmiany zachowania są zachowywane.

com.ibm.msg.client.jms.ByteStreamReadOnlyAfterSend= TRUE|FALSE
W produkcie IBM MQ 8.0.0 Fix Pack 2po wysłaniu przez aplikację komunikatu Bajty lub Strumień program IBM MQ classes for JMS może ustawić stan komunikatu, który został właśnie wysłany do tylko do odczytu lub do zapisu.

PRAWDA
Obiekty są ustawiane do odczytu tylko po wysłaniu. Ustawienie tej wartości utrzymuje kompatybilność ze specyfikacją JMS 2.0

FALSE
Obiekty są ustawiane do zapisu tylko po wysłaniu. Jest to wartość domyślna.

Pojęcia pokrewne
“Właściwość SupportMQExtensions” na stronie 324
Specyfikacja JMS 2.0 wprowadza zmiany w sposobie pracy niektórych zachowań. W produkcie IBM MQ 8.0 i nowszych znajduje się właściwość com.ibm.mq.jms.SupportMQExtensions, którą można ustawić na wartość *TRUE*, aby przywrócić te zmienione zachowania z powrotem do poprzednich implementacji.

Konfiguracja STEPLIB dla IBM MQ classes for JMS w systemie z/OS
W systemie z/OSbiblioteka STEPLIB używana w czasie wykonywania musi zawierać biblioteki IBM MQ SCSQAUTH i SCSQANLE. Te biblioteki należy określić w kodzie JCL uruchamiania lub przy użyciu pliku .profile .

Korzystając z usług UNIX and Linux System Services, można je dodać za pomocą wiersza w .profile , tak jak pokazano to w poniższym fragmencie kodu, zastępując thlqual kwalifikatorem zestawu danych wysokiego poziomu wybranym podczas instalowania produktu IBM MQ:

```
export STEPLIB=thlqual.SCSQAUTH:thlqual.SCSQANLE:$STEPLIB
```

W innych środowiskach zwykle konieczne jest zmodyfikowanie uruchamiania JCL w taki sposób, aby uwzględniał on 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 serwerem IBM MQ classes for JMS.

Wiele dużych organizacji programistycznych używa tych narzędzi do centralnego zarządzania repozytoriami bibliotek innych firm.

Plik IBM MQ classes for JMS 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 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:

- `com.ibm.mq.allclient.jar`.
- `jms.jar` jest wymagany, jeśli używany jest serwer IBM MQ classes for JMS.
- Parametr `fscontext.jar` jest wymagany, jeśli używany jest produkt IBM MQ classes for JMS i uzyskiwany jest dostęp do obiektów administrowanych JMS , które są przechowywane w kontekście JNDI systemu plików.
- `providerutil.jar` , jeśli używany jest produkt IBM MQ classes for JMS i uzyskiwany jest dostęp do obiektów administrowanych JMS , które są zapisane w kontekście JNDI systemu plików.

Od wersji IBM MQ 9.0wymagany jest dostawca zabezpieczeń Bouncy Castle i pliki JAR obsługi CMS . Więcej informacji na ten temat zawierają "Co jest zainstalowane w systemie IBM MQ classes for JMS" na stronie 87 i Wsparcie dla środowisk JRE innych niżIBM.

Uruchamianie aplikacji produktu IBM MQ classes for JMS w ramach Java security manager

Program IBM MQ classes for JMS może być uruchomiony z włączonym menedżerem zabezpieczeń produktu Java . Aby pomyślnie uruchomić aplikacje z włączoną obsługą Java security manager , należy skonfigurować produkt Java virtual machine (JVM) z odpowiednim plikiem konfiguracyjnym strategii.

Najprostszym sposobem utworzenia odpowiedniego pliku definicji strategii jest zmiana pliku konfiguracyjnego strategii dostarczanego wraz z produktem Java runtime environment (JRE). W większości systemów ten plik znajduje się w katalogu `lib/security/java.policy` w odniesieniu do katalogu JRE. Plik konfiguracyjny strategii można edytować za pomocą preferowanego edytora lub za pomocą programu narzędziowego strategii dostarczonego ze środowiskiem JRE.

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 nazwy właściwości systemowych produktu Java, o których mowa w tym temacie (**`com.ibm.mq.jms.*`**). Nie ma potrzeby zmiany żadnej istniejącej konfiguracji. Dotychczasowe nazwy właściwości systemowych również będą działać.

Jeśli używany jest mechanizm Java security manager z aplikacją, należy nadać następujące uprawnienia:

- FilePermission dla każdego pliku listy allowlist, który jest używany, z uprawnieniem do odczytu w trybie WYMUSZENIA, uprawnienia do zapisu w trybie DISCOVER.
- Właściwość PropertyPermission (odczyt) dla właściwości **`com.ibm.mq.jms.allowlist`**, **`com.ibm.mq.jms.allowlist.discover`**i **`com.ibm.mq.jms.allowlist.mode`** .

Więcej informacji na ten temat zawiera "Pojęcia dotyczące allowlistingu" na stronie 126.

Przykładowy plik konfiguracyjny strategii

Poniżej znajduje się przykład pliku konfiguracyjnego strategii, który umożliwia pomyślne uruchomienie programu IBM MQ classes for JMS w ramach domyślnego menedżera zabezpieczeń. Plik ten musi zostać dostosowany, aby określić położenia niektórych plików i katalogów: *`MQ_INSTALLATION_PATH`* reprezentuje katalog najwyższego poziomu, w którym zainstalowano produkt IBM MQ , *`MQ_DATA_DIRECTORY`* reprezentuje położenie katalogu danych MQ , a *`QM_NAME`* to nazwa menedżera kolejek, dla którego skonfigurowano 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 tym przykładzie instrukcja `grant` zawiera uprawnienia wymagane przez produkt 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 , oraz miejsca, w którym są przechowywane aplikacje.

Przykładowe aplikacje dostarczone wraz z produktem IBM MQ classes for JMSi skrypty do ich uruchomienia nie umożliwiają uruchamiania menedżera zabezpieczeń.

Konfigurowanie po instalacji dla aplikacji produktu IBM MQ classes for JMS

W tym temacie opisano, jakie aplikacje muszą być wymagane przez aplikacje IBM MQ classes for JMS w celu uzyskania dostępu do zasobów menedżera kolejek. Przedstawiono również tryby połączenia i opisano sposób konfigurowania menedżera kolejek w taki sposób, aby aplikacje mogły łączyć się 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 produkt JMS , które wymagają autoryzacji dla użytkowników nieuprzywilejowanych
Użytkownicy bez uprawnień uprzywilejowani potrzebują autoryzacji dostępu do kolejek używanych przez produkt JMS. Każda aplikacja JMS wymaga autoryzacji do menedżera kolejek, z którym działa.

Szczegółowe informacje na temat kontroli dostępu w produkcie IBM MQmożna znaleźć w sekcji Konfigurowanie zabezpieczeń.

Aplikacje produktu IBM MQ classes for JMS muszą mieć uprawnienia `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, które są używane przez obiekty MessageProducer , wymagają uprawnień `put` .
- Kolejki używane przez obiekty MessageConsumer i QueueBrowser wymagają uprawnień `get`, `inq`i `browse` .
- Metoda QueueSession.createTemporaryQueue () wymaga dostępu do kolejki modelowej określonej przez właściwość TEMPMODEL obiektu fabryki QueueConnection. Domyślną kolejką modelową jest SYSTEM.TEMP.MODEL.QUEUE.

Jeśli dowolna z tych kolejek jest kolejkami aliasami, ich kolejki docelowe wymagają uprawnień do sprawdzania uprawnień. Jeśli kolejka docelowa jest kolejką klastra, wymaga to również uprawnienia do przeglądania.

W przypadku domeny publikowania/subskrypcji, jeśli IBM MQ classes for JMS łączy się z menedżerem kolejek produktu IBM MQ w trybie migracji dostawcy przesyłania komunikatów produktu IBM MQ , używane są następujące kolejki:

- 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 przesyłania komunikatów produktu IBM MQ można znaleźć w sekcji Konfigurowanie właściwości JMS **PROVIDERVERSION** .

Dodatkowo, jeśli program IBM MQ classes for JMS łączy się z menedżerem kolejek w tym trybie, każda aplikacja, która publikuje komunikaty, potrzebuje dostępu do kolejki strumienia określonej przez fabrykę lub obiekt tematu TopicConnection. Domyślnie ta kolejka to SYSTEM.BROKER.DEFAULT.STREAM.

Jeśli używany jest produkt ConnectionConsumer, adapter zasobów produktu IBM MQ lub dostawca przesyłania komunikatów produktu WebSphere Application Server IBM MQ , może być wymagane dodatkowe autoryzacja.

Kolejki, które mają być odczytane przez parametr ConnectionConsumer , muszą mieć uprawnienia `get`, `inq`i `browse` . Systemowa kolejka niedostarczonych komunikatów oraz kolejka wycofanych komunikatów lub kolejka raportów używana przez obiekt ConnectionConsumer muszą mieć uprawnienia `put` i `passall` .

Gdy aplikacja używa trybu normalnego dostawcy przesyłania komunikatów produktu IBM MQ do przesyłania komunikatów w trybie publikowania/subskrypcji, aplikacja korzysta ze zintegrowanych funkcji

publikowania/subskrypcji udostępnianej przez menedżer kolejek. Informacje na temat zabezpieczania tematów i kolejek, które są używane, 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 rodzimego interfejsu produktu Java (JNI)” na stronie 93 .

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 “Nawiąże połączenie z menedżerem kolejek przy użyciu wywołania MQCONNX” na stronie 827

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 produktu IBM MQ classes for JMS mogły łączyć się w trybie klienta
Aby skonfigurować menedżera kolejek w taki sposób, aby aplikacje produktu IBM MQ classes for JMS mogły łączyć się w trybie klienta, należy utworzyć definicję kanału połączenia z serwerem i uruchomić proces nasłuchujący.

Tworzenie definicji kanału połączenia z serwerem

Na wszystkich platformach można użyć komendy MQSC DEFINE CHANNEL, aby utworzyć definicję kanału połączenia z serwerem. Zapoznaj się z poniższym przykładem:

```
DEFINE CHANNEL(JAVA.CHANNEL) CHLTYPE(SVRCONN) TRPTYPE(TCP)
```

IBM i W systemie IBM imożna zamiast tego użyć komendy CL CRTMQMCHL, jak w następującym 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 można użyć operacji i paneli sterujących w celu utworzenia definicji kanału połączenia z serwerem.

Nazwa kanału (JAVA.CHANNEL w poprzednich przykładach) musi być taki sam, jak nazwa kanału określona przez właściwość CHANNEL fabryki połączeń używanej przez aplikację do łączenia się z menedżerem kolejek. Wartością domyślną właściwości CHANNEL jest SYSTEM.DEF.SVRCONN.

Uruchamianie nasłuchiwania

Należy uruchomić program nasłuchujący dla menedżera kolejek, jeśli nie został on jeszcze uruchomiony.

Multi W systemie Wiele platformmożna użyć komendy MQSC START LISTENER, aby uruchomić program nasłuchujący po pierwszym 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/OSmożna użyć tylko komendy START LISTENER, tak jak w poniższym przykładzie, ale należy pamiętać, że przestrzeń adresowa inicjatora kanału musi zostać uruchomiona przed uruchomieniem nasłuchiwania:

```
START LISTENER TRPTYPE(TCP) PORT(1414)
```

IBM i W systemie IBM imożna również użyć komendy CL STRMQMLSR, aby uruchomić program nasłuchujący, tak jak w następującym przykładzie:

```
STRMQMLSR PORT(1414) MQMNAME(QMGRNAME)
```

W tej komendzie *QMGRNAME* jest nazwą menedżera kolejek.

ULW W systemie UNIX, Linux, and Windowsmożna również użyć komendy sterującej **runmqlsr** , aby uruchomić program nasłuchujący, tak jak w następującym przykładzie:

```
runmqlsr -t tcp -p 1414 -m QMgrName
```

W tej komendzie parametr *QMgrName* jest nazwą menedżera kolejek.

Windows Linux W systemach Linux i Windowsmożna również uruchomić program nasłuchujący za pomocą programu IBM MQ Explorer.

z/OS W systemie z/OSmożna również użyć operacji i paneli sterujących, aby uruchomić program nasłuchujący.

Numer portu, na którym nasłuchuje nasłuchiwanie, musi być taki sam, jak numer portu określony przez właściwość PORT fabryki połączeń używanej przez aplikację do łączenia się 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

Wraz z produktem IBM MQ classes for JMSdostarczany jest program testu sprawdzającego instalację typu punkt z punktem (IVT). 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.

Dostępny jest skrypt uruchamiający program IVT. Skrypt ma nazwę IVTRun w systemach UNIX and Linux i IVTRun.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:

```
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 klientów na wielu platformach” na stronie 1177. 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]
```

W systemach z/OS nie udostępniono równoważnego skryptu, ale można uruchomić narzędzie IVT w trybie powiązań, wywołując bezpośrednio klasę Java za pomocą następującej komendy:

```
java com.ibm.mq.jms.MQJMSIVT -nojndi [-m qmgr ] [-v providerVersion ] [-t]
```

Ścieżka klasy musi zawierać com.ibm.mqjms.jar.

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 86. 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 UNIX 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 (Uniform Resource Locator) usługi katalogowej. Adres 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 adres 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 UNIX 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” na stronie 91.
- 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 klientów na wielu platformach” na stronie 1177.

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 UNIX 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 klientów na wielu platformach” na stronie 1177 , 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(tm) 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 86. 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 UNIX 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 (Uniform Resource Locator) usługi katalogowej. Adres 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 adres 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 MQTopicConnectionoi 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 UNIX 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” na stronie 91.
- 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 klientów na wielu platformach” na stronie 1177.

Korzystanie z przykładowych aplikacji produktu IBM MQ classes for JMS

Przykładowe aplikacje produktu IBM MQ classes for JMS zawierają przegląd wspólnych funkcji interfejsu API produktu JMS . Można ich użyć do zweryfikowania konfiguracji serwera instalacji i przesyłania komunikatów oraz do pomocy przy tworzeniu własnych aplikacji.

O tym zadaniu

Jeśli potrzebna jest pomoc w tworzeniu własnych aplikacji, można użyć przykładowych aplikacji jako punktu początkowego. Zarówno źródło, jak i skompilowana wersja są udostępniane dla każdej aplikacji. Zapoznaj się z przykładowym kodem źródłowym i zidentyfikuj kroki kluczowe, aby utworzyć każdy wymagany obiekt dla aplikacji (ConnectionFactory, Connection, Session, Destination, Producer, Consumer lub both), a także aby ustawić wszystkie właściwości wymagane do określenia sposobu działania aplikacji. Więcej informacji na ten temat zawiera “Pisanie aplikacji produktu IBM MQ classes for JMS” na stronie 134. Przykłady mogą ulec zmianie w przyszłych wersjach produktu IBM MQ.

Tabela 10 na stronie 117 pokazuje, gdzie przykładowe aplikacje produktu IBM MQ classes for JMS są zainstalowane na każdej platformie:

Tabela 10. Katalogi instalacyjne dla przykładowych aplikacji produktu IBM MQ classes for JMS

Platforma	**Katalog**
UNIX UNIX 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, które zawierają co najmniej jedną aplikację przykładową, jak pokazano na Tabela 11 na stronie 117.

Tabela 11. IBM MQ classes for JMS aplikacje przykładowe

Nazwa próbki	**Opis**
JmsBrowser.java	Aplikacja przeglądarki kolejek produktu JMS , która sprawdza wszystkie dostępne komunikaty w nazwanej kolejce, bez usuwania ich, w kolejności, w jakiej byłyby odbierane przez aplikację konsumująca.
JmsConsumer.java	Aplikacja przeglądarki kolejek produktu JMS , która sprawdza wszystkie dostępne komunikaty w nazwanej kolejce, bez usuwania ich, w kolejności, w jakiej byłyby odbierane przez aplikację konsumująca, przez wyszukiwanie instancji fabryki połączeń i instancji docelowej w kontekście początkowym (ta przykład obsługuje tylko kontekst systemu plików).

Tabela 11. IBM MQ classes for JMS aplikacje przykładowe (kontynuacja)	
Nazwa próbki	**Opis**
JmsJndiConsumer.java	Aplikacja konsumująca (odbiornik lub subskrybent) produktu JMS , która odbiera komunikat z nazwanego miejsca docelowego (kolejka lub temat), wyszukując instancję fabryki połączeń i docelową instancję w kontekście początkowym (ta przykład obsługuje tylko kontekst systemu plików).
JmsJndiProducer.java	Aplikacja producenta produktu JMS (nadawca lub publikator), która wysyła prosty komunikat do nazwanego miejsca docelowego (kolejka lub temat), wyszukując instancję fabryki połączeń i docelową instancję w kontekście początkowym (ta przykład obsługuje tylko kontekst systemu plików).
JmsProducer.java	Aplikacja producenta produktu JMS (nadawca lub publikator), która wysyła prosty komunikat do nazwanego miejsca docelowego (kolejka lub temat).
/interactive/	
SampleConsumerJava.java	Odbieraj komunikaty z kolejki/kolejki.
SampleProducerJava.java	Wyślij komunikat do kolejki/kolejki.
/interactive/helper/	
BaseOptions.java	Klasa abstrakcyjna, która może zostać rozszerzona w celu udostępnienia funkcjonalności opcji użytkownika.
IsValidType.java	Klasa abstrakcyjna dla klas sprawdzania poprawności.
JmsApp.java	Klasa abstrakcyjna, która może zostać rozszerzona w celu udostępnienia funkcjonalności konsumenta/producenta.
Keys.java	Zestaw kluczy, które definiują opcje dla przykładowych aplikacji.
Literals.java	Zbiór stałych literałów.
MyContext.java	Kontekst, w którym przedstawione są opcje.
Options.java	Udostępnia funkcje dla opcji użytkownika.
OptionsPresenter.java	Kontekst, w którym prezentowane są bieżące opcje.
/simple/	
SimpleAsyncPutPTP.java	Prosta aplikacja na potrzeby przesyłania komunikatów w trybie punkt z punktem; komunikat jest wysyłany asynchronicznie (nazywany również mechanizmem przesyłania komunikatów *fire-and-forget*). Nie są odbierane żadne komunikaty.
SimpleDurableSub.java	Prosta aplikacja, która demonstruje trwałe subskrypcje.
SimpleJNDILookup.java	Minimalna i prosta aplikacja, która demonstruje wyszukiwanie obiektów JMS przy użyciu kontekstu początkowego. Nie nawiązano połączenia z menedżerem kolejek i żadne komunikaty nie są wysyłane ani odbierane.

Tabela 11. IBM MQ classes for JMS aplikacje przykładowe (kontynuacja)	
Nazwa próbki	**Opis**
SimpleMQMDRead.java	Prosta aplikacja demonstruje sposób, w jaki aplikacja produktu JMS może korzystać z pól deskryptora komunikatu produktu MQ (MQMD) jako właściwości komunikatu produktu JMS . Nie są wysyłane żadne komunikaty. Zakłada się, że używana kolejka jest zapełniana niektórymi komunikatami.
SimpleMQMDWrite.java	Prosta aplikacja demonstruje sposób, w jaki aplikacja produktu JMS może zapisywać pola deskryptora komunikatu produktu MQ (MQMD). 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-subskrypcji.
SimpleReadAheadPTP.java	Prosta aplikacja do przesyłania komunikatów w trybie punkt z punktem; komunikaty są przesyłane strumieniowo z menedżera kolejek (znane również jako narzędzie odczytu z wyprzedzeniem). Nie są wysyłane żadne komunikaty. Zakłada się, że używana kolejka jest zapełniana niektórymi komunikatami.
SimpleRequestor.java	Prosta aplikacja, która używa requestera do wysłania komunikatu żądania, a następnie oczekiwania na odpowiedź i odebrania odpowiedzi. Uwaga: Zakłada się, że inna aplikacja przetworzy komunikat żądania i wyśle komunikat odpowiedzi.
SimpleResponder.java	Prosta aplikacja, która nasłuchuje na miejscu docelowym komunikatu, a następnie wysyła odpowiedź do miejsca docelowego replyTo komunikatu. Aplikacja jest zapisywana do działania w połączeniu z przykładową SimpleRequestor .
SimpleRetainedPub.java	Prosta aplikacja, która demonstruje 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.

Produkt IBM MQ classes for JMS udostępnia skrypt o nazwie `runjms` , który może być używany do uruchamiania przykładowych aplikacji. Ten skrypt konfiguruje środowisko produktu IBM MQ , aby umożliwić uruchamianie przykładowych aplikacji produktu IBM MQ classes for JMS .

Tabela 12 na stronie 119 przedstawia położenie skryptu na każdej platformie:

Tabela 12. Położenie skryptu `runjms`	
Platforma	**Katalog**
UNIX UNIX 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`
	lub wersji
	`/qibm/proddata/mqm/java/bin/runjms64`

Tabela 12. Położenie skryptu `runjms` (kontynuacja)	
Platforma	**Katalog**
z/OS z/OS	*MQ_INSTALLATION_PATH*`java/bin/runjms`

Aby użyć skryptu `runjms` do wywołania przykładowej aplikacji, wykonaj następujące kroki:

Procedura

1. Przejdź do wiersza 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
   ```

   W przykładowej aplikacji wyświetlana jest lista 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, należy wprowadzić następujące komendy:

```
cd /opt/mqm/samp/jms/samples
/opt/mqm/java/bin/runjms JmsBrowser -m QM1 -d LQ1
```

Pojęcia pokrewne

“Co jest zainstalowane w systemie IBM MQ classes for JMS” na stronie 87
Podczas instalowania produktu IBM MQ classes for JMStworzona jest pewna liczba plików i katalogów. W systemie Windowsniektóre konfiguracje są wykonywane podczas instalacji, automatycznie ustawiając zmienne środowiskowe. Na innych platformach i w niektórych środowiskach Windows należy ustawić zmienne środowiskowe, aby możliwe było uruchamianie aplikacji produktu IBM MQ classes for JMS .

Skrypty dostarczane z produktem IBM MQ classes for JMS

Udostępniono szereg skryptów, które ułatwiają wykonywanie typowych zadań, które należy wykonać podczas korzystania z produktu IBM MQ classes for JMS.

Tabela 13 na stronie 120 zawiera listę wszystkich skryptów i ich zastosowania. Skrypty znajdują się w podkatalogu `bin` w katalogu instalacyjnym IBM MQ classes for JMS .

Tabela 13. Skrypty dostarczane z produktem IBM MQ classes for JMS	
Użyteczność	**Użyj**
Czyszczenie [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 potrzebne
DefaultConfiguration	Służy do uruchamiania domyślnej aplikacji konfiguracyjnej 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 tworzone w postaci czytelnego tekstu.

Tabela 13. Skrypty dostarczane z produktem IBM MQ classes for JMS (kontynuacja)	
Użyteczność	**Użyj**
IVTRun [1] IVTSetup [1] IVTTidy [1]	Używany w teście weryfikowania 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 109.
JMSAdmin [1]	Uruchamia narzędzie administracyjne IBM MQ JMS zgodnie z opisem w sekcji Uruchamianie narzędzia administracyjnego.
JMSAdmin.config	Plik konfiguracyjny dla narzędzia administracyjnego IBM MQ JMS , zgodnie z opisem w sekcji Konfigurowanie narzędzia administracyjnego produktu JMS.
PSIVTRun [1]	Uruchamia program testowy sprawdzający instalację publikowania/subskrypcji zgodnie z opisem w sekcji “Narzędzie do weryfikowania instalacji (IVT) publikowania/subskrypcji dla produktu IBM MQ classes for JMS” na stronie 113.
PSReportDump.class	Ta klasa jest obsługiwana w celu zachowania kompatybilności z poprzednimi wersjami, ale nie wykonuje żadnej funkcji.
setjmsenv	Ustawia zmienne środowiskowe na potrzeby uruchamiania aplikacji IBM MQ classes for JMS w 32-bitowej maszynie wirtualnej Java (JVM) w systemach UNIX and Linux , zgodnie z opisem w sekcji “Ustawianie zmiennych środowiskowych dla IBM MQ classes for JMS” na stronie 91.
setjmsenv64	Ustawia zmienne środowiskowe na potrzeby uruchamiania aplikacji IBM MQ classes for JMS w 64-bitowej maszynie JVM w systemach UNIX and Linux , zgodnie z opisem w sekcji “Ustawianie zmiennych środowiskowych dla IBM MQ classes for JMS” na stronie 91.

Uwaga:

1. W systemie Windowsnazwa pliku zawiera rozszerzenie .bat.

Obsługa środowiska OSGi

Środowisko OSGi udostępnia środowisko, które obsługuje wdrażanie aplikacji w postaci pakunków. W ramach produktu IBM MQ classes for JMSdostarczane są dziewięć pakunków OSGi.

Środowisko OSGi udostępnia ogólne, bezpieczne i zarządzane środowisko produktu Java , które obsługuje wdrażanie aplikacji, które wchodzą w postaci pakunków. Urządzenia zgodne ze środowiskiem OSGi mogą pobierać i instalować pakunki, a także usuwać je, gdy nie są już wymagane. Środowisko zarządza instalacją i aktualizacją pakunków w sposób dynamiczny i skalowalny.

Produkt 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 produktu IBM MQ dla produktu JMSzawiera sekcja Klasy IBM MQ dla architektury JMS.

com.ibm.msg.client.osgi.jms.prereq_*version_number*.jar
: Wstępnie wymagane pliki archiwum produktu 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 produktu IBM MQ w produkcie IBM MQ classes for JMS. Informacje na temat architektury warstwowej produktu IBM MQ classes for JMSmożna znaleźć w sekcji 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.osgi.allclient_*version_number*.jar
Ten plik JAR umożliwia aplikacjom korzystanie zarówno z produktów IBM MQ classes for JMS , jak i IBM MQ classes for Java, a także dołącza kod do obsługi komunikatów PCF.

com.ibm.mq.osgi.allclientprereqs_*version_number*.jar
Ten plik JAR zawiera wymagania wstępne dla produktu `com.ibm.mq.osgi.allclient_version_number.jar` , gdzie *version_number* jest numerem wersji produktu IBM MQ , który jest zainstalowany.

Pakunki są instalowane w podkatalogu `java/lib/OSGi` w instalacji produktu 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 braku możliwości uruchamiania zarówno produktu IBM MQ classes for JMS , jak i produktu IBM MQ classes for Java w ramach tych samych środowiska OSGi, jednak wszystkie pozostałe ograniczenia nadal mają zastosowanie. W przypadku wersji produktu przed IBM MQ 8.0zastosowanie ma to ograniczenie przy użyciu produktu IBM MQ classes for JMS lub IBM MQ classes for Java .

Pakunek `com.ibm.mq.osgi.java`*version_number*`.jar`, który jest również instalowany w podkatalogu `java/lib/OSGi` instalacji produktu IBM MQ lub w folderze `java\lib\OSGi` w systemie Windows, jest częścią produktu IBM MQ classes for Java. Ten pakunek nie może być załadowany do środowiska wykonawczego OSGi, w którym został załadowany produkt IBM MQ classes for JMS .

Pakunki OSGi dla produktu 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 było w stanie znaleźć wszystkie wymagane pliki DLL lub współużytkowane biblioteki.

Jeśli pakunki OSGi są używane dla produktu IBM MQ classes for JMS, tematy tymczasowe nie działają. Ponadto klasy wyjścia kanału napisane w produkcie Java nie są obsługiwane z powodu nieodłącznego problemu w ładowaniu klas w środowisku programu ładującego wiele klas, takim jak środowisko OSGi. Pakunek użytkownika może mieć informacje o pakunkach IBM MQ classes for JMS , ale pakunki produktu IBM MQ classes for JMS nie są znane w 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 można znaleźć w serwisie WWW OSGi Alliance .

z/OS V 9.1.0 MQ Adv. VUE Połączenia klienta JMS z aplikacjami wsadowymi działaniowymi w systemie z/OS

Za pomocą połączenia klienckiego aplikacja IBM MQ classes for JMS na serwerze z/OS może łączyć się z menedżerem kolejek w systemie z/OS , który zawiera atrybut **ADVCAP**(ENABLED). Korzystanie z połączenia klienckiego może uprościć topologie produktu IBM MQ .

Wartość **ADVCAP** (`WŁĄCZONE`) dotyczy tylko menedżera kolejek systemu z/OS , licencjonowanego jako IBM MQ Advanced for z/OS Value Unit Edition (patrz sekcja IBM MQ Identyfikatory produktów i informacje o eksporcie) i działającego z wartością **QMGRPROD** ustawioną na `ADVANCEDVUE`.

Patrz DISPLAY QMGR , aby uzyskać więcej informacji na temat **ADVCAP** i START QMGR , aby uzyskać więcej informacji na temat **QMGRPROD**.

Note that batch is the only environment supported; there is no support for JMS for CICS or JMS for IMS.

Aplikacja IBM MQ classes for JMS w systemie z/OS nie może korzystać z połączenia w trybie klienta z menedżerem kolejek, który nie jest uruchomiony na serwerze z/OS, lub z menedżerem kolejek, który nie ma ustawionego zestawu opcji **ADVCAP**(ENABLED) .

Jeśli aplikacja IBM MQ classes for JMS w systemie z/OS próbuje nawiązać połączenie przy użyciu trybu klienta, a nie jest to dozwolone, zostanie wygenerowany komunikat o wyjątku JMSFMQ0005 .

Obsługa Advanced Message Security (AMS)

From IBM MQ 9.1, IBM MQ classes for JMS client applications can use AMS when connecting to IBM MQ Advanced for z/OS Value Unit Edition queue managers on remote z/OS systems.

Nowy typ magazynu kluczy, `jceracfks`, jest obsługiwany tylko w systemie `keystore.conf` na platformie z/OS , 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
```

Oddzielnie otrzymywanie IBM MQ classes for JMS

IBM MQ classes for JMS są dostępne w samorozpakowujących się plikach JAR, które można pobrać z programu Fix Central , jeśli użytkownik chce uzyskać tylko pliki JAR produktu IBM MQ classes for JMS , do wdrożenia w narzędziu do zarządzania oprogramowaniem lub w celu użycia z autonomicznymi aplikacjami klienckim.

Zanim rozpoczniesz

Przed uruchomieniem tego zadania należy upewnić się, że na komputerze jest zainstalowane środowisko Java runtime environment (JRE) oraz że środowisko JRE zostało dodane do ścieżki systemowej.

Instalator produktu Java , który jest używany w tym procesie instalacji, nie wymaga działania jako użytkownik root ani żaden konkretny użytkownik. Jedynym wymaganiem jest to, że użytkownik, który jest uruchamiany, ma dostęp do zapisu w katalogu, w którym mają być zapisywane pliki.

O tym zadaniu

Przed IBM MQ 8.0, IBM WebSphere MQ classes for Java lub IBM WebSphere MQ classes for JMS nie są dostępne jako oddzielne pobieranie. W przypadku produktu IBM WebSphere MQ 7.5 lub wcześniejszego, jeśli używane są aplikacje językowe produktu Java , które korzystają z produktu IBM WebSphere MQ classes for Java lub IBM WebSphere MQ classes for JMS, należy je zainstalować, wykonując pełną

instalację serwera lub instalując jeden z klientów SupportPacs w systemie, w którym aplikacja jest rozwijana, oraz system, w którym aplikacja zostanie uruchomiona. Ta instalacja instaluje wiele plików, niż pliki IBM WebSphere MQ classes for Java i IBM WebSphere MQ classes for JMS .

Jednak w przypadku produktu IBM MQ 8.0następujące pliki są dostępne w samowydającej się pliku JAR, co pozwala zminimalizować wielkość pobierania i instalacji oraz czas wymagany do przeprowadzenia instalacji:

- IBM MQ classes for JMS
- IBM MQ classes for Java
- Adapter zasobów produktu IBM MQ
- Pakunki OSGi produktu IBM MQ

V 9.1.0.8 W produkcie IBM MQ 9.1.0 Fix Pack 8narzędzie JMSAdmin jest instalowane jako część samorozpakowujących się plików JAR, które zawierają następujące dodatkowe pliki powiązane z narzędziem JMSAdmin :

- Plik `JMSAdmin.bat` używany do uruchamiania narzędzia JMSAdmin w systemie Windows.
- Skrypt JMSAdmin, który jest używany do uruchamiania narzędzia na platformach Linux i UNIX .
- Przykładowy plik konfiguracyjny dla narzędzia JMSAdmin (`JMSAdmin.config`).

V 9.1.0.8 Klient, który jest instalowany przy użyciu samorozpakowujących się plików JAR, może użyć narzędzia JMSAdmin do tworzenia obiektów administrowanych JMS w kontekście systemu plików (plik `.bindings`). Klient może również wyszukać i użyć tych administrowanych obiektów.

V 9.1.0.8 Wcześniej samorozpakowujący się plik JAR zainstalował plik `com.ibm.mq.allclient.jar` i wszystkie jego wstępnie wymagane pliki JAR do katalogu `wmq/JavaSE` w podanym katalogu instalacyjnym. W produkcie IBM MQ 9.1.0 Fix Pack 8pliki te są instalowane w katalogu `wmq/JavaSE/lib` , a pliki powiązane z narzędziem JMSAdmin są instalowane w katalogu `wmq/JavaSE/bin` .

Po uruchomieniu wykonywalnego pliku JAR zostanie wyświetlona umowa licencyjna produktu IBM MQ , która musi zostać zaakceptowana. Prosi on o katalog, w którym należy zainstalować IBM MQ classes for Java, IBM MQ classes for JMS, adapter zasobów i pakunki OSGi. Jeśli wybrany katalog instalacyjny nie istnieje, zostanie on utworzony, a pliki programu zostaną zainstalowane. Jeśli jednak katalog istnieje, zgłaszany jest błąd i żadne pliki nie są zainstalowane.

Procedura

1. Pobierz plik JAR klienta IBM MQ Java / JMS z programu Fix Central.
   a) Kliknij ten odsyłacz: IBM MQ Java / JMS klient.
   b) W wyświetlonej liście dostępnych poprawek znajdź klienta dla używanej wersji produktu IBM MQ .
      Na przykład:

      ```
      release level: 9.1.4.0-IBM-MQ-Install-Java-All
      Continuous Delivery Release:9.1.4 IBM MQ JMS and Java 'All Client'
      ```

      Następnie kliknij nazwę pliku klienta i postępuj zgodnie z procesem pobierania.
2. Uruchom instalację z katalogu, do którego został pobrany plik.
   Aby uruchomić instalację, 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 `9.1.4.0`, a *V.R.M.F*`-IBM-MQ-Install-Java-All.jar` jest nazwą pliku, który został pobrany z programu Fix Central.

Na przykład, aby zainstalować klienta Java / JMS dla wersji IBM MQ 9.1.4 , należy użyć następującej komendy:

```
java -jar 9.1.4.0-IBM-MQ-Install-Java-All.jar
```

Uwaga: Aby przeprowadzić tę instalację, należy zainstalować na komputerze środowisko JRE i dodać je do ścieżki systemowej.

Po wprowadzeniu komendy wyświetlane są następujące informacje:

```
Zanim będzie można użyć, wyodrębnić lub zainstalować produkt IBM MQ V9.1, należy zaakceptować
warunki 1. IBM International License Agreement for Evaluation of
Programy 2. IBM International Program License Agreement i dodatkowe
informacje licencyjne. Przeczytaj uważnie poniższe umowy licencyjne.

Umowa licencyjna jest oddzielnie widziana przy użyciu
--viewLicenseOpcja umowy.

Naciśnij klawisz Enter, aby wyświetlić warunki licencji teraz, 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 od razu po wybraniu x zostanie wyświetlony następujący komunikat:

      ```
      Dodatkowe informacje o licencji są wyświetlane oddzielnie przy użyciu
      --viewLicense-opcja informacji.

      Naciśnij klawisz Enter, aby wyświetlić dodatkowe informacje licencyjne teraz, lub 'x', aby
      pominąć.
      ```

   b) Aby wyświetlić dodatkowe warunki licencji, naciśnij klawisz Enter.

      Naciśnięcie klawisza x powoduje również pominięcie wyświetlania dodatkowych warunków licencji.

      Po wyświetleniu dodatkowych warunków licencji lub od razu po wybraniu opcji 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
      zgadzają się, wybierz "Nie zgadzam się".

      Wybierz [ 1] I Zgadzam się, lub [ 2] Nie zgadzam się:
      ```

   c) Aby zaakceptować umowę licencyjną i kontynuować wybieranie katalogu instalacyjnego, wybierz 1.

      Alternatywnie, wybranie 2 powoduje natychmiastowe zakończenie instalacji.

      Jeśli zostanie wybrana wartość 1, zostanie wyświetlony następujący komunikat:

      ```
      Wprowadź katalog dla plików produktu lub pozostaw puste pole, aby zaakceptować wartość
      domyślną.
      Domyślnym katalogiem docelowym jest H: \WMQ

      Katalog docelowy dla plików produktu?
      ```

4. Określ katalog instalacyjny dla klienta Java / JMS :

   - Jeśli chcesz zainstalować pliki produktu w domyślnym położeniu, naciśnij klawisz Enter bez określania wartości.
   - Jeśli chcesz zainstalować pliki produktu w innym miejscu niż domyślne, podaj nazwę katalogu, w którym chcesz zainstalować pliki produktu, a następnie naciśnij klawisz Enter, aby rozpocząć instalację.

   Podana nazwa katalogu nie może już istnieć. W przeciwnym razie po uruchomieniu instalacji zgłaszany jest błąd i żadne pliki nie są instalowane.

   Jeśli nie istnieje, wybrany katalog instalacyjny zostanie utworzony, a pliki programu zostaną zainstalowane w tym katalogu. W trakcie instalacji w wybranym katalogu instalacyjnym zostanie utworzony nowy katalog o nazwie `wmq` . Trzy podkatalogi, `JavaEE`, `JavaSE`i `OSGi`, są tworzone w katalogu `wmq` o następującej zawartości:

```
.\JavaEE:
wmq.jmsra.ivt.ear
wmq.jmsra.rar
```

```
.\JavaSE:
com.ibm.mq.allclient.jar
com.ibm.mq.traceControl.jar
fscontext.jar
jms.jar
providerutil.jar

.\OSGi:
com.ibm.mq.osgi.allclient_V.R.M.F.jar
com.ibm.mq.osgi.allclientprereqs_V.R.M.F.jar
```

gdzie *V.R.M.F* to numer wersji, wydania, modyfikacji i pakietu poprawek.

Po zakończeniu instalacji zostanie wyświetlony komunikat z potwierdzeniem, jak pokazano w poniższym przykładzie:

```
Rozpakowywanie plików do H: \WMQ \wmq
Successfully extracted all product files.
```

Allowlisting w programie IBM MQ classes for JMS

Mechanizm przekształcania do postaci szeregowej obiektu Java i mechanizm deserializacji został zidentyfikowany jako potencjalne zagrożenie dla bezpieczeństwa. Allowlisting w programie IBM MQ classes for JMS zapewnia pewną ochronę przed pewnym ryzykiem serializacji.

Mechanizm przekształcania do postaci szeregowej i przekształcania z postaci szeregowej produktu Java został zidentyfikowany jako potencjalne ryzyko związane z bezpieczeństwem, ponieważ deserializacja tworzy instancje dowolnych obiektów produktu Java , w których istnieje możliwość złośliwego wysyłania danych w celu spowodowania różnych problemów. Jedną z nowych aplikacji serializacji jest Java Message Service (JMS) ObjectMessages , które wykorzystują serializację do hermetyzacji i przenoszenia dowolnych obiektów.

Dopuszczanie do postaci szeregowej umożliwia ograniczenie niektórych rodzajów ryzyka, które mogą być szeregowane. Jawnie określając, które klasy mogą być hermetyzowane, a wyodrębnione z ObjectMessages, allowlisting zapewnia pewną ochronę przed pewnymi zagrożeniami do serializacji.

Allowlisting w programie IBM MQ classes for JMS

Dodatkowe informacje:

- “Pojęcia dotyczące allowlistingu” na stronie 126 -przegląd opcji allowlisting
- “Konfigurowanie i korzystanie z listy allowlist JMS” na stronie 129 , aby uzyskać informacje na temat sposobu konfigurowania listy allowlist
- “Allowlisting w programie WebSphere Application Server” na stronie 132 , aby uzyskać informacje na temat sposobu konfigurowania listy allowlist w produkcie WebSphere Application Server.

Pojęcia pokrewne

“Uruchamianie aplikacji produktu IBM MQ classes for JMS w ramach Java security manager” na stronie 104

Program IBM MQ classes for JMS może być uruchomiony z włączonym menedżerem zabezpieczeń produktu Java . Aby pomyślnie uruchomić aplikacje z włączoną obsługą Java security manager , należy skonfigurować produkt Java virtual machine (JVM) z odpowiednim plikiem konfiguracyjnym strategii.

Pojęcia dotyczące allowlistingu

W produkcie IBM MQ classes for JMSobsługa dopuszczania klas w implementacji interfejsu JMS ObjectMessage umożliwia ograniczenie niektórych zagrożeń związanych z bezpieczeństwem, które mogą być związane z serializacją obiektu Java i mechanizmem deserializacji.

Allowlisting w programie IBM MQ classes for JMS

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 nazwy właściwości

systemowych produktu Java, o których mowa w tym temacie (**com.ibm.mq.jms.***). Nie ma potrzeby zmiany żadnej istniejącej konfiguracji. Dotychczasowe nazwy właściwości systemowych również będą działać.

Produkt IBM MQ classes for JMS obsługuje wyświetlanie listy klas w implementacji interfejsu JMS ObjectMessage .

Lista allowlist definiuje, które klasy produktu Java mogą być przekształcane do postaci szeregowej za pomocą ObjectMessage.setObject() i zdeserializowane z parametrem ObjectMessage.getObject().

Próby przekształcenia do postaci szeregowej lub przekształcenia z postaci szeregowej instancji klasy, która nie znajduje się na liście allowlist z elementem ObjectMessage , powodują zgłoszenie wyjątku javax.jms.MessageFormatException z java.io.InvalidClassException jako jego przyczyną.

Tworzenie listy allowlist

Ważne: Produkt IBM MQ classes for JMS nie może być dystrybuowany z listą allowlist. Wybór klas, które mają zostać przesłane za pomocą ObjectMessages , jest wyborem projektu aplikacji, a program IBM MQ nie może go zaołabować.

Z tego powodu mechanizm allowlistingu pozwala na dwa tryby pracy:

Wykrywanie
W tym trybie mechanizm generuje listę pełnych nazw klas, raportyjąc wszystkie klasy, które zostały zaobserwowane do postaci szeregowej lub zesterializowane w obszarze ObjectMessages.

Egzekwowanie
W tym trybie mechanizm wymusza zezwalanie na dopuszczanie, odrzucanie prób serializowania lub deserializowania klas, które nie znajdują się na liście allowlist.

Jeśli chcesz użyć tego mechanizmu, musisz najpierw uruchomić w trybie DISCOVERY, aby zebrać listę aktualnie serializowanych i zdeserializowanych klas, przejrzeć listę i użyć go jako podstawy dla listy allowlist. Korzystanie z listy może być nawet niezmienione, ale lista musi zostać najpierw przejrzana, zanim zdecydujesz się na to.

Sterowanie mechanizmem allowlistingu

Dostępne są trzy właściwości systemowe umożliwiające sterowanie mechanizmem allowlistingu:

com.ibm.mq.jms.allowlist
Tę właściwość można określić w jeden z następujących sposobów:

- Nazwa ścieżki do pliku, który zawiera listę allowlist, w formacie identyfikatora URI pliku (czyli począwszy od `file:`). W trybie DISCOVERY plik ten jest zapisywany w mechanizmie allowlistingowym. Plik nie może istnieć. Jeśli plik istnieje, mechanizm zgłasza wyjątek, a nie nadpisuje go. W trybie WYMUSZENIE ten plik jest odczytyany przez mechanizm allowlistingu.
- Rozdzielana przecinkami pełna nazwa klasy, która stanowi listę allowlist.

Jeśli ta właściwość nie jest ustawiona, mechanizm listy allowlist jest nieaktywny.

Jeśli używany jest produkt Java security manager, należy upewnić się, że pliki JAR produktu IBM MQ classes for JMS mają dostęp do odczytu i zapisu do tego pliku.

com.ibm.mq.jms.allowlist.discover

- Jeśli ta właściwość nie jest ustawiona lub ma wartość false, mechanizm listy allowlist działa w trybie WYMUSZENIE.
- Jeśli ta właściwość jest ustawiona na wartość true, a lista allowlist została określona jako identyfikator URI pliku, mechanizm listy allowlist działa w trybie DISCOVERY.
- Jeśli ta właściwość jest ustawiona na wartość true, a lista allowlist została określona jako lista nazw klas, to mechanizm listy allowlist zgłasza odpowiedni wyjątek.
- Jeśli ta właściwość jest ustawiona na wartość true, a lista allowlist nie została określona przy użyciu właściwości com.ibm.mq.jms.allowlist, mechanizm listy allowlist jest nieaktywny.

- Jeśli ta właściwość jest ustawiona na wartość true, a plik allowlist już istnieje, mechanizm listy allowlist zgłasza wyjątek java.io.InvalidClassException , a wpisy nie są dodawane do pliku.

com.ibm.mq.jms.allowlist.mode
Tę właściwość łańcucha można określić na jeden z trzech sposobów:

- Jeśli ta właściwość jest ustawiona na wartość SERIALIZE, to tryb WYMUSZENIA wykonuje sprawdzanie poprawności listy allowlist tylko w metodzie ObjectMessage.setObject().
- Jeśli ta właściwość jest ustawiona na wartość DESERIALIZE, to tryb WYMUSZENIA wykonuje sprawdzanie poprawności listy allowlist tylko w metodzie ObjectMessage.getObject().
- Jeśli ta właściwość jest nieustawiona lub ustawiona na inną wartość, tryb WYMUSZENIA wykonuje sprawdzanie poprawności listy dozwolonych dla metod ObjectMessage.getObject() i ObjectMessage.setObject().

Format pliku allowlist

Są to główne cechy formatu pliku allowlist:

- Plik allowlist jest domyślnym kodowaniem plików platformy z końcami linii właściwymi dla platformy.

  Uwaga: Jeśli używany jest plik allowlist, plik ten jest zawsze zapisywany i odczytyowany przy użyciu domyślnego kodowania plików dla maszyny JVM.

  Jest to w porządku, jeśli plik allowlist jest generowany w jeden z następujących sposobów:

  – z/OS Wygenerowane przez autonomiczną aplikację działającą na serwerze z/OS i używane przez inne aplikacje autonomiczne, które również działają w systemie z/OS.
  – Wygenerowane przez aplikację działającą w produkcie WebSphere Application Server na dowolnej platformie i używane przez inną instancję produktu WebSphere Application Server.
  – Multi Generowane przez autonomiczną aplikację działającą na serwerze IBM MQ for Multiplatformsi używane przez inne aplikacje autonomiczne działające na serwerze IBM MQ for Multiplatformslub przez aplikacje działające wewnątrz produktu WebSphere Application Server na dowolnej platformie.

  Jednak ponieważ produkt WebSphere Application Server korzysta ze standardu ASCII, a autonomiczna maszyna JVM używa kodu EBCDIC, problemy z kodowaniem plików będą mieć miejsce, jeśli plik allowlist zostanie wygenerowany w jeden z następujących sposobów:

  – Generowany w systemie z/OS, a następnie używany przez aplikacje autonomiczne działające na platformie innej niż z/OS lub przez produkt WebSphere Application Server.
  – Wygenerowany przez produkt WebSphere Application Server lub aplikację autonomiczną działającą na platformie innej niż z/OS, a następnie używany przez autonomiczną aplikację w systemie z/OS.
- Każda niepusta linia zawiera pełną nazwę klasy. Puste wiersze są ignorowane.
- Komentarze można dołączać-wszystko po znaku '#', na końcu wiersza, jest ignorowane.
- Istnieje bardzo podstawowy mechanizm dzikiej przyrody:

  – Element '*' może być elementem **ostatniego** nazwy klasy.
  – Znak '*' jest zgodny z **pojedynczym** elementem nazwy klasy, to znaczy klasy, ale nie ma części pakietu.

  Oznacza to, że `com.ibm.mq.*` pasuje do `com.ibm.mq.MQMessage` , ale nie `com.ibm.mq.jmqi.remote.api.RemoteFAP`.

  Znaki wieloznaczne nie działają dla klas w pakiecie domyślnym, który jest przeznaczony dla klas bez jawnej nazwy pakietu, dlatego nazwa klasy "*" jest odrzucana.
- Źle sformatowane pliki allowlist, na przykład pliki zawierające wpis, taki jak `com.ibm.mq.*.Message`, gdzie znak wieloznaczny nie jest ostatnim elementem, powodują zgłoszenie wyjątku java.lang.IllegalArgumentException .
- Pusty plik allowlist ma wpływ na całkowite wyłączenie użycia obiektu ObjectMessage.

Format listy allowlist w postaci listy rozdzielanej przecinkami

Ten sam mechanizm tworzenia znaków wieloznacznych jest dostępny dla listy allowlist, która jest listą rozdzielaną przecinkami.

- Znak '*' może być rozwijany przez system operacyjny, jeśli jest określony w wierszu komend lub w skrypcie powłoki lub w pliku wsadowym, dlatego może wymagać specjalnej obsługi.
- Znak komentarza '#' ma zastosowanie tylko wtedy, gdy określony jest plik. Jeśli lista allowlist jest określona jako rozdzielona przecinkami lista nazw klas, to zakładając, że system operacyjny lub powłoka nie przetworzy go, ponieważ jest to domyślny znak komentarza w wielu powłokach UNIX lub Linux , jest traktowany jako normalny znak.

Kiedy dozwolone jest wyświetlanie listy allowlistingu?

Allowlisting jest inicjowany, gdy aplikacja po raz pierwszy uruchamia metodę ObjectMessage setMessage() lub getMessage().

Właściwości systemowe są wartościowane, plik allowlist jest otwierany i w trybie wymuszania, lista klas allowlisty jest ładowana po zainicjowaniu mechanizmu. W tym momencie wpis jest zapisywany w pliku dziennika programu IBM MQ JMS dla aplikacji.

Gdy mechanizm jest inicjowany, jego parametry mogą nie zostać zmienione. Ponieważ czas inicjowania nie jest łatwo przewidywany, ponieważ zależy on od zachowania aplikacji. Ustawienia właściwości systemu i zawartość pliku allowlist powinny być zatem traktowane jako ustalone od czasu uruchomienia aplikacji. Nie należy zmieniać właściwości ani zawartości pliku allowlist w czasie, gdy aplikacja jest uruchomiona, ponieważ wyniki nie są gwarantowane.

Punkty do rozważenia

Najlepszym podejściem do minimalizowania ryzyka związanego z mechanizmem przekształcania do postaci szeregowej produktu Java byłoby zbadanie alternatywnych metod przesyłania danych, takich jak format JSON zamiast ObjectMessage. Korzystanie z mechanizmów Advanced Message Security (AMS) umożliwia dodawanie kolejnych zabezpieczeń poprzez zapewnienie, że komunikaty pochodzą z zaufanych źródeł.

Jeśli używany jest mechanizm Java security manager z aplikacją, należy nadać następujące uprawnienia:

- FilePermission dla każdego pliku listy allowlist, który jest używany, z uprawnieniem do odczytu w trybie WYMUSZENIA, uprawnienia do zapisu w trybie DISCOVER.
- Właściwość PropertyPermission (odczyt) dla właściwości **com.ibm.mq.jms.allowlist**, **com.ibm.mq.jms.allowlist.discover**i **com.ibm.mq.jms.allowlist.mode** .

Więcej informacji

Więcej informacji na temat allowlist można znaleźć w sekcji “Konfigurowanie i korzystanie z listy allowlist JMS” na stronie 129 i “Allowlisting w programie WebSphere Application Server” na stronie 132 .

Pojęcia pokrewne

“Uruchamianie aplikacji produktu IBM MQ classes for JMS w ramach Java security manager” na stronie 104

Program IBM MQ classes for JMS może być uruchomiony z włączonym menedżerem zabezpieczeń produktu Java . Aby pomyślnie uruchomić aplikacje z włączoną obsługą Java security manager , należy skonfigurować produkt Java virtual machine (JVM) z odpowiednim plikiem konfiguracyjnym strategii.

Konfigurowanie i korzystanie z listy allowlist JMS

Informacje te informują użytkownika o tym, jak działa lista allowlist i jak można je skonfigurować, korzystając z funkcji zawartych w IBM MQ classes for JMS w celu wygenerowania pliku allowlist, zawierającego listę typów 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 nazwy właściwości systemowych produktu Java, o których mowa w tym temacie (**com.ibm.mq.jms.***). Nie ma potrzeby zmiany żadnej istniejącej konfiguracji. Dotychczasowe nazwy właściwości systemowych również będą działać.

Przed uruchomieniem tego zadania upewnij się, że przeczytałeś i zrozumiałeś “Pojęcia dotyczące allowlistingu” na stronie 126

O tym zadaniu

Jeśli włączono funkcję allowlistingu, produkt IBM MQ classes for JMS korzysta z tej funkcji w następujący sposób:

- Gdy aplikacja chce wysłać obiekt ObjectMessage, może on utworzyć go na jeden z dwóch sposobów:
  - Metoda Session.createObjectMessage(Serializable), przechodząc do obiektu, który ma być zawarty w komunikacie.
  - Metoda Session.createObjectMessage() umożliwia utworzenie pustego obiektu ObjectMessage, a następnie wywołanie metody ObjectMessage.setObject(Serializable) w celu zapisania obiektu, który ma zostać wysłany w obrębie obiektu ObjectMessage.

  Gdy wywoływane są metody Session.createObjectMessage(Serializable) lub ObjectMessage.setObject(Serializable), klasy dla usługi JMS sprawdzają, czy przekazany obiekt jest typu, który jest wymieniony na liście allowlist.

  Jeśli jest to typ wymieniony, obiekt jest przekształcany do postaci szeregowej i zapisywany w obrębie obiektu ObjectMessage. Jeśli jednak obiekt jest typu, którego nie ma na liście allowlist, IBM MQ classes for JMS zgłasza wyjątek JMSException zawierający komunikat:

  ```
  JMSCC0052: Wystąpił wyjątek podczas serializowania obiektu:
  'java.io.InvalidClassException: < klasa_obiektu>; Klasa nie może być serializowana
  lub z deserializacją, ponieważ nie została uwzględniona na liście allowlist '< allowlist>'.
  ```

  z powrotem do aplikacji.

  Ważne: Jeśli wyjątek jest zgłaszany przy użyciu metody Session.createObjectMessage(Serializable), obiekt ObjectMessage nie zostanie utworzony. Podobnie, jeśli wyjątek JMSException jest zgłaszany z poziomu metody ObjectMessage.setObject(Serializable), obiekt nie zostanie dodany do obiektu ObjectMessage.
- Jeśli aplikacja otrzyma obiekt ObjectMessage, wywołuje metodę ObjectMessage.getObject(), aby pobrać obiekt w nim zawarty. 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 allowlist.

  Jeśli tak, obiekt jest deserializowany i zwracany do aplikacji. Jeśli jednak obiekt jest typu, którego nie ma na liście allowlist, IBM MQ classes for JMS zgłasza wyjątek JMSException zawierający komunikat:

  ```
  JMSCC0053: Wystąpił wyjątek podczas deserializacji komunikatu:
  'java.io.InvalidClassException: < klasa_obiektu >; Klasa może nie być
  serializowane lub zerializowane, ponieważ nie zostały uwzględnione w
  allowlist '< allowlist>'. '.
  ```

  z powrotem do aplikacji.

Załóżmy na przykład, że aplikacja zawiera następujący kod, aby wysłać obiekt ObjectMessage zawierający obiekt o typie 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 jest w stanie pomyślnie umieścić komunikat w wymaganym miejscu docelowym.

Jeśli zostanie utworzony plik o nazwie `C:\allowlist.txt` zawierający pojedynczą pozycję `java.net.URL`, a następnie zostanie uruchomiony ponownie aplikację z zestawem właściwości systemu Java:

```
-Dcom.ibm.mq.jms.allowlist=file:/C:/allowlist.txt
```

Funkcja allowlist jest włączona. Aplikacja nadal jest w stanie utworzyć i wysłać obiekt ObjectMessage zawierający obiekt typu java.net.URI , ponieważ ten typ jest określony na liście allowlist.

Jeśli jednak plik `allowlist.txt` zostanie zmieniony w taki sposób, że plik zawiera pojedynczą pozycję `java.util.Calendar`, ponieważ funkcja listy allowlist jest nadal włączona, podczas wywołań aplikacji:

```
ObjectMessage msg = session.createObjectMessage(testURL);
```

IBM MQ classes for JMS sprawdź listę allowlist i sprawdź, czy nie zawiera ona wpisu dla klasy java.net.URI.

W związku z tym zgłaszany jest wyjątek JMSException zawierający komunikat JMSCC0052 .

Podobnie, załóżmy, że istnieje inna aplikacja, która otrzymuje ObjectMessages za pomocą 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 jest w stanie odbierać obiekty ObjectMessages , które zawierają obiekt dowolnego typu. Następnie aplikacja sprawdza, czy obiekt jest typu java.net.URL przed wykonaniem odpowiedniego przetwarzania.

Jeśli aplikacja jest teraz uruchomiona z właściwością systemową Java :

```
-Dcom.ibm.mq.jms.allowlist=java.net.URL
```

ustaw, funkcja allowlistingu jest włączona. Gdy aplikacja wywołuje:

```
Object messageBody = objectMessage.getObject();
```

Metoda ObjectMessage.getObject() zwraca tylko obiekty o typie java.net.URL.

Jeśli obiekt znajdujący się w obiekcie ObjectMessage nie jest tego typu, metoda ObjectMessage.getObject() zgłasza wyjątek JMSException zawierający komunikat JMSCC0053 . Następnie aplikacja musi zdecydować, co należy zrobić z komunikatem. Na przykład komunikat może zostać przeniesiony do kolejki niedostarczonych komunikatów dla tego menedżera kolejek.

Aplikacja zwraca tylko normalnie, jeśli obiekt znajdujący się wewnątrz obiektu ObjectMessage ma typ java.net.URL.

Procedura

1. Uruchom aplikację, która przetwarza ObjectMessagesz podanymi 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 program IBM MQ classes for JMS tworzy plik, który zawierał typy obiektów, które przetwarzana jest aplikacja.
2. Po przetworzeniu przez aplikację reprezentatywnej próbki ObjectMessages w określonym czasie zatrzymaj ją.

Plik allowlist zawiera teraz listę wszystkich typów obiektów zawartych w obiekcie ObjectMessages przetwarzanego przez aplikację w czasie jego działania.

Jeśli aplikacja została uruchomiona przez wystarczającą ilość czasu, lista ta zawiera wszystkie możliwe typy obiektów zawartych w obiekcie ObjectMessages , które mogą być obsługiwane przez aplikację.

3. Zrestartuj aplikację, korzystając z następującego zestawu właściwości systemowych:

```
-Dcom.ibm.mq.jms.allowlist=file:/<path to your allowlist file>
```

Dzięki temu możliwe jest włączenie opcji allowlisting, a jeśli program IBM MQ classes for JMS wykryje element ObjectMessage typu, który nie znajduje się na liście allowlist, zostanie zgłoszony wyjątek JMSException zawierający komunikat JMSCC0052 lub JMSCC0053 .

Allowlisting w programie WebSphere Application Server

Sposób użycia opcji IBM MQ classes for JMS allowlisting w programie 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 nazwy właściwości systemowych produktu Java, o których mowa w tym temacie (**`com.ibm.mq.jms.*`**). Nie ma potrzeby zmiany żadnej istniejącej konfiguracji. Dotychczasowe nazwy właściwości systemowych również będą działać.

Należy upewnić się, że instalacja produktu WebSphere Application Server obejmuje wersję adaptera zasobów produktu IBM MQ , który obsługuje opcję allowlisting.

Więcej informacji na temat korzystania z tych dwóch produktów można znaleźć w sekcji “Używanie produktów IBM MQ i WebSphere Application Server razem” na stronie 506 .

Począwszy od wersji IBM MQ 9.0.0 Fix Pack 1 , należy uwzględnić odpowiednią funkcjonalność.

Po zaktualizowaniu serwera aplikacji można użyć właściwości systemu Java :

- `-Dcom.ibm.mq.jms.allowlist`
- `-Dcom.ibm.mq.jms.allowlist.discover`

opisane w sekcji “Konfigurowanie i korzystanie z listy allowlist JMS” na stronie 129.

Uwaga: Należy ustawić właściwości systemowe produktu Java jako ogólne argumenty wirtualnej maszyny języka Java, na Java virtual machine używanym do uruchamiania serwera aplikacji, a serwer aplikacji zrestartowany w celu uwzględnienia zmian.

Więcej informacji na ten temat zawiera sekcja *Ogólne argumenty maszyny JVM* w sekcji Ustawienia maszyny wirtualnejJava .

Aby ustawić właściwości, należy przejść do okna Java virtual machine w sekcji *Definicje procesów* i wprowadzić odpowiedni argument.

Następujące ustawienie:

```
-Dcom.ibm.mq.jms.allowlist=<youruserId>_MyObject
```

Powoduje, że serwer aplikacji używa listy allowlist *youruserId_MyObject*. Tylko obiekty 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 używany był tryb *Wykryj* , a szczegóły rekordu JMS ObjectMessages, który jest przetwarzany przez serwer aplikacji, do pliku `C:\allowlist.txt`

Następujące ustawienie:

```
-Dcom.ibm.mq.jms.allowlist=file:C/:allowlist.txt
```

Powoduje, że serwer aplikacji ładuje plik `C:/allowlist.txt`i używa informacji znajdujących się w tym pliku w celu określenia listy allowlist.

Pojęcia pokrewne

“Uruchamianie aplikacji produktu IBM MQ classes for JMS w ramach Java security manager” na stronie 104

Program IBM MQ classes for JMS może być uruchomiony z włączonym menedżerem zabezpieczeń produktu Java . Aby pomyślnie uruchomić aplikacje z włączoną obsługą Java security manager , należy skonfigurować produkt Java virtual machine (JVM) z odpowiednim plikiem konfiguracyjnym strategii.

Konwersje łańcucha znaków w programie IBM MQ classes for JMS

Opcje CharsetEncoders i CharsetDecoders produktu IBM MQ classes for JMS są używane bezpośrednio do konwersji łańcuchów znaków. Domyślne zachowanie konwersji łańcucha znaków można skonfigurować przy użyciu dwóch właściwości systemowych. Obsługa komunikatów, które zawierają znaki bez odwzorowania, można skonfigurować za pomocą właściwości komunikatu w celu ustawienia działania UnmappableCharacteri bajtów zastępczych.

Przed programem IBM MQ 8.0konwersje łańcuchów w produkcie IBM MQ classes for JMS zostały wykonane przez wywołanie metod `java.nio.charset.Charset.decode(ByteBuffer)` i `Charset.encode(CharBuffer)` .

Użycie jednej z tych metod spowoduje, że zostanie użyta wartość domyślna (REPLACE) zniekształcona lub nieprzetłumaczalne dane. To zachowanie może spowodować błędy w aplikacjach i prowadzić do wystąpienia nieoczekiwanych znaków, na przykład ?, w przetłumaczonych danych.

Aby produkt IBM MQ 8.0mógł wcześniej i skuteczniej wykrywać takie problemy, należy IBM MQ classes for JMS użyć komendy CharsetEncoders i CharsetDecoders bezpośrednio i jawnie skonfigurować obsługę zniekształcanych i nieprzetłumaczalnych danych. Domyślnym zachowaniem jest REPORT takie problemy, zgłaszając odpowiedni `MQException`.

Konfigurowanie

Translacja z UTF-16 (reprezentacja znakowa używana w produkcie Java) do rodzimego zestawu znaków, na przykład UTF-8, jest termed `encoding`, podczas gdy translacja w przeciwnym kierunku jest termed `decoding`.

Obecnie dekodowanie odbywa się domyślnie w przypadku produktu `CharsetDecoders`, zgłaszając błędy, zgłaszając wyjątek.

Jedno ustawienie służy do określenia `java.nio.charset.CodingErrorAction` , aby sterować obsługą błędów przy kodowaniu i dekodowaniu. Jedno z pozostałych ustawień jest używane do sterowania bajtem zastępczym lub bajtami, gdy kodowanie jest używane. Domyślny łańcuch zastępujący Java będzie używany w operacjach dekodowania.

UnmappableCharacter-ustawienia działań i bajtów zastępczych w klasach IBM MQ dla JMS

W produkcie IBM MQ 8.0dostępne są dwie dodatkowe właściwości służące do ustawiania działania UnmappableCharacteri bajtów zastępczych. Odpowiednie definicje stałe znajdują się w katalogu `com.ibm.msg.client.wmq.WMQConstants` .

JMS_IBM_UNMAPPABLE_ACTION

Ustawia lub pobiera wartość `CodingErrorAction` , która ma być stosowana, gdy znak nie może być odwzorowany w operacji kodowania lub dekodowania.

Należy ustawić tę wartość jako `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 zastępcze, które mają być stosowane, gdy znak nie może być odwzorowany w operacji kodowania.

Domyślny łańcuch zastępujący Java jest używany w operacjach dekodowania.

```
public static final String JMS_IBM_UNMAPPABLE_REPLACEMENT = "JMS_IBM_Unmappable_Replacement";
```

Właściwości `JMS_IBM_UNMAPPABLE_ACTION` i `JMS_IBM_UNMAPPABLE_REPLACEMENT` mogą być ustawione dla miejsc docelowych lub komunikatów. Wartość ustawiona w komunikacie przesłania wartość ustawioną w miejscu docelowym, do którego wysyłany jest komunikat.

Należy zauważyć, że parametr `JMS_IBM_UNMAPPABLE_REPLACEMENT` musi być ustawiony jako pojedynczy bajt.

Właściwości systemowe służące do ustawiania wartości domyślnych systemu

W produkcie IBM MQ 8.0dostępne są następujące dwie właściwości systemowe produktu Java w celu skonfigurowania domyślnego zachowania w odniesieniu do konwersji łańcuchów znaków.

com.ibm.mq.cfg.jmqi.UnmappableCharacterAction
Określa działanie, które ma zostać podjęte w przypadku nieprzetłumaczalnych danych na temat kodowania i dekodowania. Wartością może być `REPORT`, `REPLACE`lub `IGNORE`.

com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement
Ustawia lub pobiera bajty zastępcze, które mają być stosowane, gdy znak nie może być odwzorowany w operacji kodowania. Domyślny łańcuch zastępujący Java jest używany podczas dekodowania operacji.

Aby uniknąć nieporozumień między znakami Java a reprezentacjami bajtowymi, należy podać `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, podczas gdy jest 111, jeśli rodzimy zestaw znaków jest ustawiony na EBCDIC.

Uwaga: Należy zwrócić uwagę, że jeśli obiekt MQMD lub MQMessage ma ustawione pola **`unmappableAction`** lub **`unMappableReplacement`** , wówczas wartości tych pól mają pierwszeństwo przed właściwościami systemu Java . Pozwala to na nadpisanie wartości określonych przez właściwości systemowe Java dla każdego komunikatu, jeśli jest to wymagane.

Pojęcia pokrewne
“Konwersje łańcucha znaków w programie IBM MQ classes for Java” na stronie 347
Opcje CharsetEncoders i CharsetDecoders produktu IBM MQ classes for Java są używane bezpośrednio do konwersji łańcuchów znaków. Domyślne zachowanie konwersji łańcucha znaków można skonfigurować przy użyciu dwóch właściwości systemowych. Obsługa komunikatów zawierających znaki niemożliwe do odwzorowania może być skonfigurowana za pomocą programu `com.ibm.mq.MQMD`.

Pisanie aplikacji produktu IBM MQ classes for JMS

Po krótkim wprowadzeniu do modelu JMS ten temat zawiera szczegółowe wskazówki dotyczące sposobu pisania aplikacji IBM MQ classes for JMS .

Model JMS

Model JMS definiuje zestaw interfejsów, które mogą być używane przez aplikacje produktu Java do wykonywania operacji przesyłania komunikatów. IBM MQ classes for JMS, jako dostawca JMS , definiuje sposób, w jaki obiekty JMS są powiązane z pojęciami IBM MQ . Specyfikacja JMS oczekuje, że niektóre

obiekty JMS będą administrowane obiektami. Produkt JMS 2.0 wprowadza uproszczony interfejs API, zachowując jednocześnie klasyczny interfejs API z produktu JMS 1.1.

Specyfikacja JMS oraz pakiet javax.jms definiują zestaw interfejsów, które mogą być używane przez aplikacje produktu Java do wykonywania operacji przesyłania komunikatów.

W produkcie IBM MQ 8.0produkt IBM MQ obsługuje wersję JMS 2.0 standardu JMS, która wprowadza uproszczony interfejs API, zachowując jednocześnie klasyczny interfejs API z produktu JMS 1.1.

Uproszczony interfejs API

Produkt JMS 2.0 wprowadza uproszczony interfejs API, zachowując jednocześnie specyficzne dla domeny interfejsy niezależne od produktu JMS 1.1. Uproszczona funkcja API redukuje liczbę obiektów potrzebnych do wysyłania i odbierania komunikatów, a także składa się z następujących interfejsów:

ConnectionFactory
Obiekt ConnectionFactory jest administrowany obiektem, 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. Obiekty kontekstu produktu JMSmogą być tworzone z innych obiektów kontekstu produktu JMS, a bazowe połączenie jest duplikowane.

JMSProducent
Producent produktu JMSjest tworzony przy użyciu kontekstu produktu JMSi służy do wysyłania komunikatów do kolejki lub tematu. Obiekt producenta JMSpowoduje utworzenie obiektów, które są wymagane do wysłania komunikatu.

JMSKonsument
Konsument produktu JMSjest tworzony przy użyciu kontekstu produktu JMSi służy do odbierania komunikatów z tematu lub kolejki.

Uproszczony interfejs API ma szereg efektów:

- Obiekt kontekstu JMSzawsze automatycznie uruchamia bazowe połączenie.
- Producenci JMSi konsumenci produktu JMSmogą teraz pracować bezpośrednio z ciałami komunikatów, bez konieczności pobierania całego obiektu komunikatu przy użyciu metody getBody komunikatu.
- Właściwości komunikatu można ustawić w obiekcie producenta produktu JMSprzy użyciu łańcucha łączenia metod przed wysłaniem treści wiadomości. Producent produktu JMSbędzie obsługiwać tworzenie wszystkich obiektów, które są potrzebne do wysłania komunikatu. Za pomocą produktu JMS 2.0można ustawić właściwości, a komunikat jest wysyłany w następujący sposób:

```
context.createProducer().
setProperty("foo", "bar").
setTimeToLive(10000).
setDeliveryMode(NON_PERSISTENT).
setDisableMessageTimestamp(true).
send(dataQueue, body);
```

Produkt JMS 2.0 wprowadza również subskrypcje współużytkowane, w których komunikaty mogą być współużytkowane przez wielu konsumentów. Wszystkie subskrypcje produktu JMS 1.1 są traktowane jako subskrypcje niewspółużytkowane.

Klasyczne API

Poniższa lista zawiera podsumowanie głównych interfejsów produktu JMS w klasycznym interfejsie API:

Miejsce docelowe
Miejsce docelowe to miejsce, w którym aplikacja wysyła komunikaty lub jest to źródło, z którego aplikacja odbiera komunikaty, lub oba te komunikaty.

ConnectionFactory
Obiekt ConnectionFactory hermetyzuje zestaw właściwości konfiguracyjnych dla połączenia. Aplikacja korzysta z fabryki połączeń w celu utworzenia połączenia.

Połączenie
Obiekt połączenia hermetyzuje aktywne połączenie aplikacji z serwerem przesyłania komunikatów. Aplikacja korzysta z połączenia w celu utworzenia sesji.

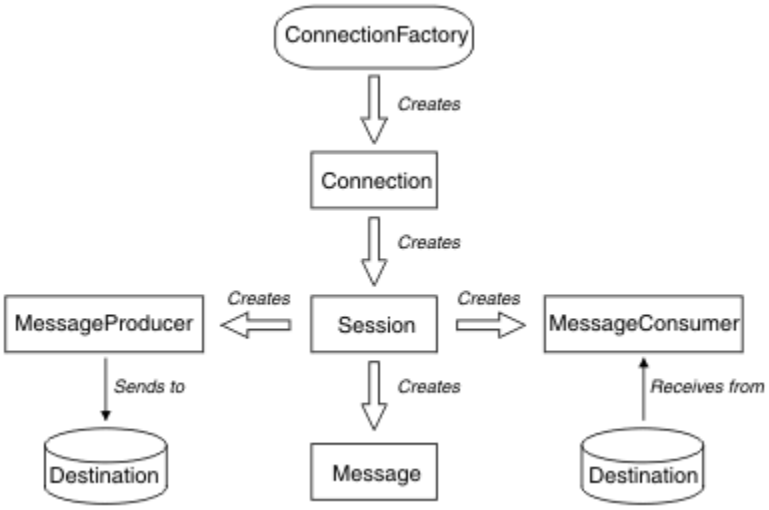
Sesja
Sesja jest jednowątkowym kontekstem do wysyłania i odbierania komunikatów. Aplikacja korzysta z sesji w celu utworzenia komunikatów, producentów komunikatów i konsumentów komunikatów. Sesja jest transakcowana lub nie jest transakowana.

Komunikat
Obiekt komunikatu hermetyzuje komunikat, który aplikacja wysyła lub odbiera.

MessageProducer
Aplikacja korzysta z producenta komunikatów w celu wysyłania komunikatów do miejsca docelowego.

MessageConsumer
Aplikacja używa konsumenta komunikatów do odbierania komunikatów wysyłanych do miejsca docelowego.

Rysunek 9 na stronie 136 przedstawia te obiekty i ich relacje.



Rysunek 9. Obiekty JMS i ich relacje

Obiekt docelowy, obiekt ConnectionFactorylub obiekt połączenia może być używany współbieżnie przez różne wątki aplikacji wielowątkowej, ale nie można jednocześnie używać obiektu Session, MessageProducerlub obiektu MessageConsumer przez różne wątki. Najprostszym sposobem zapewnienia, że obiekt Session, MessageProducerlub obiekt MessageConsumer nie jest używany współbieżnie, jest utworzenie osobnego obiektu sesji dla każdego wątku.

Produkt JMS obsługuje dwa style przesyłania komunikatów:

- Przesyłanie komunikatów w modelu punkt-punkt
- Przesyłanie komunikatów publikowania/subskrypcji

Te style przesyłania komunikatów są również nazywane *domenami przesyłania komunikatów*, a użytkownik może łączyć oba style przesyłania komunikatów w aplikacji. W domenie typu punkt z punktem miejsce docelowe jest kolejką, a w domenie publikowania/subskrybowania-miejscem docelowym jest temat.

W przypadku wersji produktu JMS przed JMS 1.1, programowanie dla domeny typu punkt z punktem wykorzystuje jeden zestaw interfejsów i metod, a programowanie dla domeny publikowania/subskrybowania używa innego zestawu. Oba zestawy są podobne, ale oddzielne. W produkcie JMS 1.1można używać wspólnego zestawu interfejsów i metod obsługujących zarówno domeny

przesyłania komunikatów. Wspólne interfejsy udostępniają niezależny widok domeny dla każdej domeny przesyłania komunikatów. Tabela 14 na stronie 137 zawiera listę niezależnych interfejsów domeny JMS i odpowiadających im interfejsów specyficznych dla domeny.

Tabela 14. Niezależne interfejsy domeny JMS i specyficzne dla domeny

Niezależne interfejsy domeny	**Interfejsy specyficzne dla domeny dla domeny punkt z punktem**	**Interfejsy specyficzne dla domeny dla domeny publikowania/subskrybowania**
ConnectionFactory	Fabryka QueueConnection	Fabryka TopicConnection
Połączenie	QueueConnection	TopicConnection
Miejsce docelowe	Kolejka	Temat
Sesja	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver QueueBrowser	TopicSubscriber

Program JMS 2.0 zachowuje wszystkie interfejsy specyficzne dla domeny, a więc istniejące aplikacje nadal mogą używać tych interfejsów. Jednak w przypadku nowych aplikacji należy rozważyć użycie niezależnych interfejsów domeny produktu JMS 1.1 lub uproszczonego interfejsu API produktu JMS 2.0.

W produkcie IBM MQ classes for JMSobiekty produktu JMS są powiązane z pojęciami produktu IBM MQ w następujący sposób:

- Obiekt połączenia ma właściwości, które są pobierane z właściwości fabryki połączeń, która została użyta do utworzenia połączenia. Te właściwości sterują sposobem łączenia się aplikacji z menedżerem kolejek. Przykładami tych właściwości są nazwa menedżera kolejek oraz, dla aplikacji, która łączy się z menedżerem kolejek w trybie klienta, nazwa hosta lub adres IP systemu, na którym jest uruchomiony menedżer kolejek.
- Obiekt sesji hermetyzuje uchwyt połączenia IBM MQ , który w związku z tym definiuje zasięg transakcyjny sesji.
- Obiekt MessageProducer i obiekt MessageConsumer każdy hermetyzuje uchwyt obiektu IBM MQ .

W przypadku korzystania z produktu IBM MQ classes for JMSstosowane są wszystkie normalne reguły produktu IBM MQ . Należy zwrócić uwagę, 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 mają być administrowane obiektami. Administrator tworzy i obsługuje administrowane obiekty w centralnym repozytorium, a aplikacja JMS pobiera te obiekty za pomocą interfejsu JNDI (Java Naming and Directory Interface).

W programie IBM MQ classes for JMSimplementacja interfejsu docelowego jest abstrakcyjną nadklasą kolejki i tematu, a więc instancją miejsca docelowego jest obiekt kolejki lub obiekt tematu. Niezależne interfejsy domeny traktują kolejkę lub temat jako miejsce docelowe. Domena przesyłania komunikatów dla obiektu MessageProducer lub MessageConsumer jest określana na podstawie tego, czy miejscem docelowym jest kolejka, czy temat.

Dlatego w programie IBM MQ classes for JMS obiekty następujących typów mogą być administrowanymi obiektami:

- ConnectionFactory
- Fabryka QueueConnection
- Fabryka TopicConnection
- Kolejka

- Temat
- XAConnectionFactory
- Fabryka XAQueueConnection
- Fabryka XATopicConnection

Pojęcia pokrewne

Interfejsy językowe programu IBM MQ Java

“Korzystanie z funkcji produktu JMS 2.0” na stronie 319

Produkt JMS 2.0 wprowadza kilka nowych obszarów funkcjonalności do produktu IBM MQ classes for JMS.

Komunikaty produktu JMS

Komunikaty produktu JMS składają się z nagłówka, właściwości i treści. JMS definiuje pięć typów treści komunikatu.

Komunikaty produktu JMS składają się z następujących części:

Nagłówek
Wszystkie komunikaty obsługują ten sam zestaw pól nagłówka. Pola nagłówka zawierają wartości, które są używane przez klientów i dostawców do identyfikowania i kierowania komunikatów.

Właściwości
Każdy komunikat zawiera wbudowane narzędzie do obsługi wartości właściwości zdefiniowanych przez aplikację. Właściwości udostępniają efektywny mechanizm filtrowania komunikatów zdefiniowanych przez aplikację.

Treść
Produkt JMS definiuje pięć typów treści komunikatu, które obejmują większość stylów przesyłania komunikatów, które są obecnie używane:

Strumień
Strumień wartości podstawowych Java . Jest on wypełniany i odczytywaniany sekwencyjnie.

Odwzoruj
Zestaw par nazwa-wartość, gdzie nazwy są łańcuchami, a wartości są typami podstawowymi Java . Dostęp do pozycji można uzyskać w sposób sekwencyjny lub losowy według nazwy. Kolejność pozycji jest niezdefiniowana.

Tekstowy
Komunikat zawierający obiekt java.lang.String.

Obiekt
Komunikat zawierający obiekt Java z możliwością serializacji

Bajty
Strumień niewysłanych bajtów. Ten typ komunikatu dotyczy dosłownie kodowania treści w celu dopasowania do istniejącego formatu komunikatu.

Pole nagłówka JMSCorrelationID służy do połączenia jednego komunikatu z innym komunikatem. Zwykle łączy ona komunikat odpowiedzi z żądaniem komunikatu. Atrybut JMSCorrelationID może zawierać identyfikator komunikatu specyficznego dla dostawcy, łańcuch specyficzny dla aplikacji lub wartość bajtu dostawcy [] w formacie dostawcy.

Selektory komunikatów w produkcie JMS

Komunikaty mogą zawierać wartości właściwości zdefiniowane przez aplikację. Aplikacja może używać selektorów komunikatów do posiadania komunikatów filtru dostawcy produktu JMS .

Komunikat zawiera wbudowane narzędzie do obsługi wartości właściwości zdefiniowanych przez aplikację. Ten mechanizm udostępnia mechanizm dodawania pól nagłówka specyficznego dla aplikacji do komunikatu. Właściwości umożliwiają aplikacji, przy użyciu selektorów komunikatów, możliwość wyboru lub filtrowania komunikatów dostawcy produktu JMS w jego imieniu przy użyciu kryteriów specyficznych dla aplikacji. Właściwości zdefiniowane przez aplikację muszą być przestrzegane następujących reguł:

- Nazwy właściwości muszą być przestrzegane w regułach dla identyfikatora selektora komunikatów.

- Wartości właściwości mogą być typu Boolean, byte, short, int, long, float, double i String.
- Przedrostki JMSX i JMS_ name są zastrzeżone.

Wartości właściwości są ustawiane przed wysłaniem komunikatu. Gdy klient otrzymuje komunikat, właściwości komunikatu są tylko do odczytu. Jeśli klient podejmie próbę ustawienia właściwości w tym momencie, zgłaszany jest wyjątek MessageNotWriteableException . Jeśli wywoływane jest pole clearProperties , wówczas właściwości mogą być odczytywanych zarówno z, jak i zapisywane w.

Wartość właściwości może zduplikować wartość w treści komunikatu. Produkt JMS nie definiuje strategii dla elementów, które mogą zostać wprowadzone do właściwości. Programiści aplikacji muszą jednak mieć świadomość, że dostawcy produktu JMS prawdopodobnie obsługują dane w treści komunikatu bardziej wydajnie niż dane we właściwościach komunikatu. Aby uzyskać najlepszą wydajność, aplikacje muszą używać właściwości komunikatu tylko wtedy, gdy muszą dostosować nagłówek komunikatu. Podstawową przyczyną tego działania jest obsługa dostosowanej selekcji komunikatów.

Selektor komunikatów produktu JMS umożliwia klientowi określenie komunikatów, które są zainteresowane przy użyciu nagłówka komunikatu. Dostarczane są tylko komunikaty z nagłówkami, które są zgodne z selektorem.

Selektory komunikatów nie mogą odwoływać się do wartości treści komunikatu.

Selektor komunikatów jest zgodny z komunikatem, gdy selektor ma wartość true, gdy pole nagłówka komunikatu i wartości właściwości są podstawiane dla odpowiadających im identyfikatorów w selektorze.

Selektor komunikatów to łańcuch, którego składnia jest oparta na podzbiorze składni wyrażeń warunkowych SQL92 . Kolejność, w jakiej selektor komunikatów jest wartościowany, jest od lewej do prawej w obrębie poziomu priorytetu. Aby zmienić to zamówienie, można użyć nawiasów. Predefiniowane literały selektora i nazwy operatorów są zapisywane w tym miejscu wielkimi literami, jednak nie są one w nich rozróżniane.

Zawartość selektora komunikatów

Selektor komunikatów może zawierać:

- Literały
  - Literał łańcuchowy jest ujęty w znaki cudzysłowu. Podwojony znak cudzysłowu reprezentuje cudzysłów. Przykładami są literały i literał. Podobnie jak w przypadku literałów łańcuchowych Java , używane są kodowanie znaków Unicode.
  - Dokładny literał liczbowy jest wartością liczbową bez przecinka dziesiętnego, np. 57, -957 i +62. Obsługiwane są liczby z zakresu od Java .
  - Przybliżony literał liczbowy to wartość liczbowa w notacji naukowej, taka jak 7E3 lub -57.9E2, lub wartość liczbowa z dziesiętnym, takim jak 7., -95.7lub +6.2. Obsługiwane są liczby z zakresu Java double.
  - Literały boolowskie TRUE i FALSE.
- Identyfikatory:
  - Identyfikator jest nieograniczoną sekwencją znaków Java liter i cyfr Java , z których pierwsza musi być literą Java . Litera to dowolny znak, dla którego metoda Character.isJavaLetter zwraca wartość true. Obejmuje to _ oraz $. 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 lub IS.
  - Identyfikatory są odwołaniami do pól nagłówka lub odwołaniami do właściwości.
  - W identyfikatorach rozróżniana jest wielkość liter.
  - Odwołania do pól nagłówka komunikatu są ograniczone do:
    - JMSDeliveryMode
    - JMSPriority

- JMSMessageID
- JMSTimestamp
- JMSCorrelationID
- JMSType

Wartości JMSMessageID, JMSTimestamp, JMSCorrelationIDi JMSType mogą mieć wartość NULL, a jeśli tak, są traktowane jako wartość NULL.

– Każda nazwa rozpoczynający się od znaku JMSX jest nazwą właściwości zdefiniowaną w produkcie JMS.

– Każda nazwa rozpoczynający się od JMS_ jest nazwą właściwości specyficzną dla dostawcy.

– Każda nazwa, która nie zaczyna się od JMS , jest nazwą właściwości specyficznej dla aplikacji. Jeśli istnieje odwołanie do właściwości, która nie istnieje w komunikacie, jej wartość jest równa NULL. Jeśli ta wartość istnieje, jej wartością jest odpowiadająca jej wartość właściwości.

- Biały znak jest taki sam, jak w przypadku produktu Java: obszar, karta pozioma, kanał informacyjny formularza i terminator linii.
- Wyrażenia:
  – Selektor jest wyrażeniem warunkowym. Selektor wartościowany do prawdziwych dopasowań; selektor, którego wartościowanie ma wartość false lub nieznany, nie jest zgodny.
  – Wyrażenia arytmetyczne składają się z samych siebie, operacji arytmetycznych, identyfikatorów (z wartością, która jest traktowana jako literał liczbowy), oraz literałów liczbowych.
  – Wyrażenia warunkowe składają się z samych siebie, operacji porównania i operacji logicznych.
- Standard bracketing (), aby ustawić kolejność, w jakiej wyrażenia są wartościowane, jest obsługiwane.
- Operatory logiczne w kolejności kolejności wykonywania: NOT, AND, OR.
- Operatory porównania: =, >, > =, <, < =, < > (nie równe).
  – Porównywane są tylko wartości tego samego typu. Jedynym wyjątkiem jest to, że poprawne jest porównanie dokładnych wartości liczbowych i przybliżonych wartości liczbowych. (Wymagana konwersja typów jest zdefiniowana przez reguły promocji numerycznej produktu Java). Jeśli istnieje próba porównania różnych typów, selektor zawsze ma wartość false.
  – Porównywanie łańcuchowe i boolowskie jest ograniczone do = i < >. Dwa łańcuchy są równe tylko wtedy, gdy zawierają taką samą sekwencję znaków.
- Operatory arytmetyczne w kolejności kolejności wykonywania zadań:
  – +,-unary.
  – *,/, mnożenie i dzielenie.
  – +,-,-, dodawanie i odejmowanie.
  – Operacje arytmetyczne na wartości NULL nie są obsługiwane. Jeśli są one uruchomione, pełny selektor zawsze ma wartość false.
  – Operacje arytmetyczne muszą używać promocji numerycznej Java .
- Operator porównania arithmetic-expr1 [NOT] BETWEEN arithmetic-expr2 i arithmetic-expr3 :
  – Wiek od 15 do 19 lat jest odpowiednikiem wieku > = 15 AND wiek < = 19.
  – Wiek NIE MIĘDZY 15 a 19 jest odpowiednikiem wieku < 15 LUB wieku > 19.
  – Jeśli którekolwiek z wyrażeń operacji BETWEEN ma wartość NULL, wartością tej operacji jest false (fałsz). Jeśli którekolwiek z wyrażeń operacji NOT BETWEEN ma wartość NULL, wartość tej operacji jest równa true.
- identyfikator [NOT] IN (string-literal1, string-literal2, ...) operator porównania, gdzie identyfikator ma wartość String lub NULL.
  – Kraj IN ("Wielka Brytania", "USA", "Francja") ma wartość "true" dla "UK" i "false" dla "Peru". Jest on równoważny z wyrażeniem (Country = 'UK ') OR (Country = 'US') OR (Country = 'Francja ').

  - Kraj NOT IN ("UK", "US", "Francja") ma wartość "false" dla "UK" i "true" w odniesieniu do "Peru". Jest ono równoznaczne z wyrażeniem NOT ((Country = 'UK ') OR (Country = 'US') OR (Country = 'France ')).
  - Jeśli identyfikator operacji IN lub NOT IN ma wartość NULL, wartość tej operacji jest nieznana.
- identifier [NOT] LIKE wzorzec-wartość [ESCAPE escape-character] operator porównania, gdzie identyfikator ma wartość łańcuchową. wzorzec-wartość jest literałem łańcuchowym, gdzie _ oznacza dowolny pojedynczy znak, a% oznacza dowolną sekwencję znaków (włącznie z pustą sekwencją). Wszystkie inne postacie stoją dla siebie. Opcjonalny znak zmiany znaczenia jest jednoznakowym literałem łańcuchowym o znaku, który jest używany do zmiany znaczenia specjalnego znaczenia _ i% w wartości wzorca.
  - Telefon LIKE '12%3' ma wartość true dla 123 i 12993, a wartość false dla 1234.
  - Słowo LIKE 'l_se' ma wartość true dla "lose" i false dla "loose".
  - podkreślono wartość LIKE '_ %' ESCAPE' \ ' jest wartością true dla opcji "_foo" i false dla "bar".
  - Telefon NOT LIKE '12%3' ma wartość false dla 123 i 12993, a wartość true dla 1234.
  - Jeśli identyfikator operacji LIKE lub NOT LIKE ma wartość NULL, wartość tej operacji jest nieznana.
- identyfikator IS NULL operator porównania testów dla wartości pola nagłówka o wartości NULL lub brakującej wartości właściwości.
  - prop_name ma wartość NULL.
- Identyfikator IS NOT NULL operator porównania testów dla istnienia wartości pola nagłówka innej niż NULL lub wartości właściwości.
  - prop_name IS NOT NULL.

Przykład selektora komunikatów

Poniższy selektor komunikatów wybiera komunikaty z typem komunikatu samochodu, koloru niebieskiego i wagą większą niż 2500 funtów:

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

Wartości właściwości NULL

Jak zauważono na poprzedniej liście, wartości właściwości mogą mieć wartość NULL. Wartościowanie wyrażeń selektora, które zawierają wartości NULL, jest definiowane przez semantykę NULL języka SQL 92. Poniższa lista zawiera krótki opis tych semantyki:

- SQL traktuje wartość NULL jako nieznaną.
- Porównanie lub arytmetyka z nieznaną wartością zawsze daje nieznaną wartość.
- Operator IS NULL przekształca nieznaną wartość w wartość PRAWDA.
- Operator IS NOT NULL przekształca nieznaną wartość w wartość FALSE.

Specjalne zachowanie właściwości JMSMessageID i JMSCorrelationID

Klasy IBM MQ classes for JMS zawierają optymalizacje podczas wybierania komunikatów z kolejki na podstawie wartości JMSMessageID lub JMSCorrelationID.

Jeśli aplikacja określa selektor formularza, wykonaj następujące czynności:

JMSMessageID= 'ID:*id_komunikatu*'

gdzie *id_komunikatu* to łańcuch zawierający standardowy identyfikator komunikatu produktu IBM MQ , a klasy IBM MQ dla usługi JMS używają identyfikatora MQMO_MATCH_MSG_ID produktu **MatchOption** w celu pobrania komunikatu z określonym identyfikatorem komunikatu.

Aby na przykład uzyskać komunikat z identyfikatorem komunikatu 414D51207061756C745639314C545320C57C1A5F25ECE602 z kolejki, aplikacja powinna użyć następującego selektora komunikatów:

```
JMSMessageID='ID:414D51207061756C745639314C545320C57C1A5F25ECE602'
```

Podobnie, jeśli aplikacja określa selektor, który ma format:

JMSCorrelationID = 'Identyfikator:*correlation_id*'

gdzie *correlation_id* to łańcuch zawierający standardowy identyfikator korelacji IBM MQ , klasy IBM MQ classes for JMS używają obiektu **MatchOption** MQMO_MATCH_CORREL_ID w celu pobrania komunikatu z określonym identyfikatorem korelacji z kolejki.

W poniższym przykładzie selektor komunikatów jest używany do pobierania komunikatu z identyfikatorem korelacji 414D51207061756C745639314C545320846E5B5F25B1CC02:

```
JMSCorrelationID='ID:414D51207061756C745639314C545320846E5B5F25B1CC02'
```

Jeśli selektor komunikatów zawiera wartość wszystkich zer dla identyfikatora *message_id* lub *correlation_id*, to jest on zgodny z dowolnym komunikatem w kolejce. Na przykład, jeśli aplikacja korzysta z selektora:

```
JMSMessageID='ID:000000000000000000000000000000000000000000000000'
```

wtedy każdy komunikat w kolejce jest uznawany za zgodny i zwrócony do aplikacji.

Więcej informacji na temat wartości 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 są dostępne. Dlatego dokładne literały liczbowe są ograniczone do tych bez liczby dziesiętnej. Dlatego też istnieją cyfry z separatą dziesiętną jako alternatywna reprezentacja dla przybliżonej wartości liczbowej.

Komentarze SQL nie są obsługiwane.

Odwzorowywanie komunikatów produktu JMS na komunikaty produktu IBM MQ
Komunikaty produktu IBM MQ składają się z deskryptora komunikatu, opcjonalnego nagłówka MQRFH2 i treści. Treść komunikatu produktu JMS jest częściowo odwzorowywana i częściowo kopiowana do komunikatu produktu IBM MQ .

W tym temacie opisano, w jaki sposób struktura komunikatu produktu JMS opisana w pierwszej części tej sekcji jest odwzorowywana na komunikat IBM MQ . Interesują go programiści, którzy chcą przesyłać komunikaty między produktem JMS a tradycyjnymi aplikacjami produktu IBM MQ . Interesują go także osoby, które chcą manipulować komunikatami przesyłanych między dwiema aplikacjami JMS , na przykład w implementacji IBM Integration Bus .

Ta sekcja nie ma zastosowania, jeśli aplikacja korzysta z połączenia w czasie rzeczywistym z brokerem. Jeśli aplikacja korzysta z połączenia w czasie rzeczywistym, cała komunikacja jest wykonywana bezpośrednio za pośrednictwem protokołu TCP/IP; nie są związane żadne kolejki ani komunikaty produktu IBM MQ .

Komunikaty produktu IBM MQ składają się z trzech komponentów:

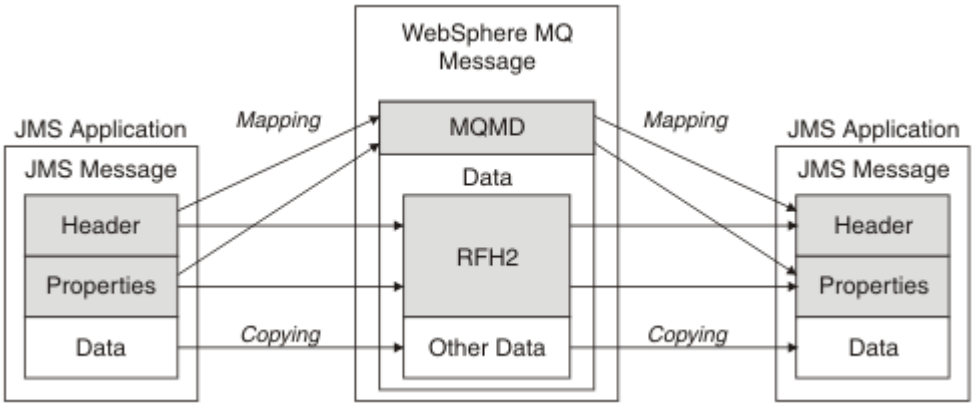
- Deskryptor komunikatu produktu IBM MQ (MQMD)
- Nagłówek IBM MQ MQRFH2
- Treść komunikatu.

Parametr 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 MQRFH2 zawiera informacje specyficzne dla produktu JMS,

zawsze należy je uwzględnić w komunikacie, gdy nadawca wie, że miejscem docelowym odbioru jest aplikacja JMS . W normalnych warunkach pomiń MQRFH2 podczas wysyłania komunikatu bezpośrednio do aplikacji innej niżJMS . Jest to spowodowane tym, że taka aplikacja nie oczekuje MQRFH2 w jego komunikacie IBM MQ .

Jeśli komunikat przychodzący nie ma nagłówka MQRFH2 , to obiekt kolejki lub tematu uzyskany z pola nagłówka JMSReplyTo komunikatu domyślnie ma ten zestaw flag, dzięki czemu komunikat odpowiedzi wysłany do kolejki lub tematu również nie ma nagłówka MQRFH2 . Można wyłączyć to zachowanie, w tym nagłówek MQRFH2 w komunikacie odpowiedzi tylko wtedy, gdy oryginalny komunikat ma nagłówek MQRFH2 , ustawiając właściwość TARGCLIENTMATCHING dla fabryki połączeń na wartość NO.

Rysunek 10 na stronie 143 pokazuje, w jaki sposób struktura komunikatu produktu JMS jest transformowana do komunikatu produktu IBM MQ i z powrotem:



Rysunek 10. Sposób transformowania komunikatów między JMS i IBM MQ przy użyciu nagłówka MQRFH2

Struktury są transformowane na dwa sposoby:

Odwzorowanie
Jeśli deskryptor MQMD zawiera pole równoważne z polem JMS , pole JMS jest odwzorowywane na pole MQMD. Dodatkowe pola MQMD są prezentowane jako właściwości produktu JMS , ponieważ aplikacja JMS może wymagać pobrania lub ustawienia tych pól podczas komunikowania się z aplikacją inną niżJMS .

Kopiowanie
Jeśli nie ma odpowiednika MQMD, pole lub właściwość nagłówka JMS jest przekazywana, prawdopodobnie transformowana, jako pole wewnątrz MQRFH2.

Nagłówek MQRFH2 i JMS
W tej kolekcji tematów opisano nagłówek MQRFH, wersja 2, który zawiera dane specyficzne dla produktu JMS, które są powiązane z treścią komunikatu. Nagłówek MQRFH w wersji 2 jest rozszerzalny, a ponadto może zawierać dodatkowe informacje, które nie są bezpośrednio powiązane z produktem JMS. Ta sekcja obejmuje jednak tylko jej użycie przez produkt JMS.Pełny opis znajduje się w sekcji MQRFH2 -Reguły i formatowanie nagłówka 2.

Znajdują się tam dwie części nagłówka, część stała i część zmiennej.

Stała część
Stała część jest modelowana na podstawie wzorca nagłówka *standard* IBM MQ 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 zdefiniowane.

Wersja (MQLONG)
Numer wersji struktury.

Musi to być wartość MQRFH_VERSION_2 (wartość: 2) (wartość początkowa).

StrucLength (MQLONG)
Łączna długość MQRFH2, w tym pola danych NameValue.

Wartość ustawiona w polu StrucLength musi być wielokrotnością 4 (dane w polach danych NameValuemogą być dopełniane znakami spacji, aby to osiągnąć).

Kodowanie (MQLONG)
Kodowanie danych.

Kodowanie dowolnych danych liczbowych w części komunikatu po wykonaniu komendy MQRFH2 (następny nagłówek lub dane komunikatu po tym nagłówku).

CodedCharSetId (MQLONG)
Identyfikator kodowanego zestawu znaków.

Reprezentacja dowolnych danych znakowych w części komunikatu po MQRFH2 (następny nagłówek lub dane komunikatu po tym nagłówku).

Format (MQCHAR8)
Nazwa formatu.

Nazwa formatu dla części komunikatu, która jest następująca po MQRFH2.

Flagi (MQLONG)
Flagi.

MQRFH_NO_FLAGS = 0. Nie ustawiono flag.

NameValueCCSID (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, które są zawarte w nagłówku (StrucID i Format).

Jeśli identyfikator CCSID NameValueto 2-bajtowy identyfikator CCSID Unicode (1200, 13488 lub 17584), kolejność bajtów w kodzie Unicode jest taka sama, jak kolejność bajtów w polach liczbowych w tabeli MQRFH2. (Na przykład sam identyfikator CCSID wersji, StrucLengthi NameValue).

Identyfikator CCSID NameValueprzyjmuje wartości z następującej tabeli:

Tabela 15. Możliwe wartości dla pola identyfikatora CCSID NameValue

CCSID	**Znaczenie**
1200	UTF-16, najnowsza obsługiwana wersja Unicode
13488	UTF-16, podzbiór Unicode w wersji 2.0
17584	UTF-16, podzbiór Unicode w wersji 3.0 (zawiera symbol Euro)
1208	UTF-8, najnowsza obsługiwana wersja Unicode

Część zmiennej
Część zmiennej jest zgodna ze stałą porcją. Część zmiennej zawiera zmienną liczbę folderów MQRFH2 . Każdy folder zawiera zmienną liczbę elementów lub właściwości. Właściwości powiązane z grupą folderów. Nagłówki MQRFH2 utworzone przez produkt JMS mogą zawierać dowolne 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 16. 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 17. 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 18. 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.

W programie Tabela 19 na stronie 146 jest wyświetlana pełna lista nazw właściwości.

Tabela 19. Foldery MQRFH2 i właściwości używane przez produkt JMS				
JMS nazwa pola	**Java typ**	**Nazwa folderu MQRFH2**	**Nazwa właściwości**	**Typ/wartości**
JMSDestination	Miejsce docelowe	jms	Dst	string (łańcuch)
JMSExpiration	long	jms	Wyg.	i8
JMSPriority	int	jms	PRI	i4
JMSDeliveryMode	int	jms	Dlv	i4
JMSCorrelationID	Łańcuch	jms	CID	string (łańcuch)
JMSReplyTo	Miejsce docelowe	jms	Rto	string (łańcuch)
JMSTimestamp	long	jms	tms	i8
JMSType	Łańcuch	MCD	Typ, Ustaw, Fmt	string (łańcuch)
JMSXGroupID	Łańcuch	jms	Identyfikator grupy	string (łańcuch)
JMSXGroupSeq	int	jms	Sekw	i4
xxx (zdefiniowana przez użytkownika)	Dowolna	USR	xxx	dowolne

Tabela 19. Foldery MQRFH2 i właściwości używane przez produkt JMS (kontynuacja)				
JMS nazwa pola	**Java typ**	**Nazwa folderu MQRFH2**	**Nazwa właściwości**	**Typ/wartości**
		MCD	MSD	jms_none tekst_jms jms_bytes jms_map Strumień jms_ obiekt jms_object

NameValue(długość nazwy) (MQLONG)
Długość (w bajtach) łańcucha danych NameValuewystępujący bezpośrednio po tej zmiennej długości (nie obejmuje własnej długości).

Dane NameValue(MQCHARn)
Pojedynczy łańcuch znaków, którego długość w bajtach jest podana w poprzedzającym polu NameValueLength. Zawiera on folder zawierający sekwencję właściwości. Każda właściwość jest tercetu typu nazwa/typ/wartość, zawartego w elemencie XML, którego nazwa jest nazwą folderu, w następujący sposób:

```
<foldername>
triplet1 triplet2 .....  tripletn </foldername>
```

Po zamykaniu znacznika `</foldername>` mogą występować spacje jako znaki dopełniania. Każdy triplet jest zakodowany przy użyciu składni typu XML:

```
<name dt='datatype'>value</name>
```

Element `dt='datatype'` jest opcjonalny i jest pomijany dla wielu właściwości, ponieważ typ danych jest predefiniowany. Jeśli ta opcja jest włączona, przed znacznikiem `dt=` musi zostać umieszczony jeden lub więcej znaków spacji.

name
jest nazwą właściwości; patrz Tabela 19 na stronie 146.

datatype
musi być zgodne, po złożeniu, jeden z typów danych wymienionych w Tabela 20 na stronie 147.

value
jest łańcuchową reprezentacją wartości, która ma być transportowana, przy użyciu definicji w produkcie Tabela 20 na stronie 147.

Wartość NULL jest zakodowana przy użyciu następującej składni:

```
<name dt='datatype' xsi:nil='true'></name>
```

Nie należy używać produktu `xsi:nil='false'`.

Tabela 20. Typy danych właściwości	
Typ danych	**Definicja**
string (łańcuch)	Dowolna sekwencja znaków z wyjątkiem < i &
boolean (boolowskie)	Znak `0` lub `1` (`0` = false, `1` = true)

Tabela 20. Typy danych właściwości (kontynuacja)	
Typ danych	**Definicja**
bin.hex	Cyfry szesnastkowe reprezentujące oktety
i1	Liczba, wyrażona za pomocą cyfr `0..9`, z opcjonalnym znakiem (bez ułamków lub wykładników). Musi leżeć w zakresie od -128 do 127 włącznie
i2	Liczba, wyrażona za pomocą cyfr `0..9`, z opcjonalnym znakiem (bez ułamków lub wykładników). Musi leżeć w zakresie od -32768 do 32767 włącznie
i4	Liczba, wyrażona za pomocą cyfr `0..9`, z opcjonalnym znakiem (bez ułamków lub wykładników). Musi leżeć w zakresie od -2147483648 do 2147483647 włącznie
i8	Liczba, wyrażona za pomocą cyfr `0..9`, z opcjonalnym znakiem (bez ułamków lub wykładników). Musi leżeć w zakresie od -9223372036854775808 do 9223372036854775807 włącznie
int	Liczba, wyrażona za pomocą cyfr `0..9`, z opcjonalnym znakiem (bez ułamków lub wykładników). Musi znajdować się w tym samym zakresie co i8. Może być używany w miejsce jednego z typów i *, jeśli nadawca nie chce skojarzyć konkretnej precyzji z właściwością
r4	Liczba zmiennopozycyjna, wielkość < = 3.40282347E+38, > = 1.175E-37 wyrażona przy użyciu cyfr `0..9`, opcjonalny znak, opcjonalne cyfry ułamkowe, opcjonalny wykładnik
r8	Liczba zmiennopozycyjna, wielkość < = 1.7976931348623E+308, > = 2.225E-307 wyrażona przy użyciu 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 produktu JMS z odpowiednimi polami MQMD
W tych tabelach są wyświetlane pola MQMD równoważne z polami nagłówka JMS , właściwościami JMS i właściwościami specyficznymi dla dostawcy JMS .

Tabela 21 na stronie 148 zawiera listę pól nagłówka JMS , a Tabela 22 na stronie 149 zawiera listę właściwości JMS , które są odwzorowywane bezpośrednio na pola MQMD. Tabela 23 na stronie 149 zawiera listę właściwości specyficznych dla dostawcy oraz pól MQMD, do których są one odwzorowane.

Tabela 21. 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

Tabela 21. Odwzorowanie pól nagłówka JMS na pola MQMD (kontynuacja)			
JMS Pole nagłówka	**Java typ**	**Pole MQMD**	**Typ C**
JMSMessageID	Łańcuch	MsgID	MQBYTE24
JMSTimestamp	long	PutDate PutTime	MQCHAR8 MQCHAR8
JMSCorrelationID	Łańcuch	CorrelId	MQBYTE24

Tabela 22. Odwzorowanie właściwości produktu JMS na pola MQMD			
JMS property	**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 23. Odwzorowanie właściwości specyficznych dla dostawcy produktu JMS na pola MQMD			
Właściwość specyficzna dla dostawcy JMS	**Java typ**	**Pole MQMD**	**Typ C**
Wyjątek JMS_IBM_Report_Exception	int	Raport	MQLONG
Wygaśnięcie JMS_IBM_Report_Expiration	int	Raport	MQLONG
Plik JMS_IBM_Report_COA	int	Raport	MQLONG
JMS_IBM_Report_COD	int	Raport	MQLONG
Plik JMS_IBM_Report_PAN	int	Raport	MQLONG
Plik JMS_IBM_Report_NAN	int	Raport	MQLONG
JMS_IBM_Report_Pass_Msg_ID	int	Raport	MQLONG
JMS_IBM_Report_Pass_Correl_ID	int	Raport	MQLONG
JMS_IBM_Report_Discard_Msg	int	Raport	MQLONG
JMS_IBM_MsgType	int	MsgType	MQLONG
Opinie JMS_IBM_Feedback	int	Opinie	MQLONG
Format JMS_IBM_Format	Łańcuch	Format “1” na stronie 150	MQCHAR8
Typ JMS_IBM_PutAppl	int	Typ_aplikacji_wstawiającej	MQLONG
JMS_IBM_Encoding	int	Kodowanie	MQLONG

Tabela 23. Odwzorowanie właściwości specyficznych dla dostawcy produktu JMS na pola MQMD (kontynuacja)			
Właściwość specyficzna dla dostawcy JMS	**Java typ**	**Pole MQMD**	**Typ C**
Zestaw znaków JMS_IBM_Character_Set	Łańcuch	CodedCharacterSetId “2” na stronie 150	MQLONG
JMS_IBM_PutDate	Łańcuch	PutDate	MQCHAR8
JMS_IBM_PutTime	Łańcuch	PutTime	MQCHAR8
Grupa JMS_IBM_Last_Msg_In_Group	boolean (boolowskie)	MsgFlags	MQLONG

Uwaga:

1. Format JMS_IBM_Format reprezentuje format treści komunikatu. Może to być zdefiniowane przez aplikację ustawiające właściwość JMS_IBM_Format komunikatu (należy zauważyć, że istnieje limit 8 znaków) lub może być domyślnie ustawiona na format IBM MQ treści komunikatu odpowiedniej dla typu komunikatu JMS . Pliki JMS_IBM_Format są odwzorowywać na pole Format MQMD tylko wtedy, gdy komunikat nie zawiera sekcji RFH lub RFH2 . W typowym komunikacie jest on odwzorowyany na pole Format w RFH2 bezpośrednio poprzedzające treść komunikatu.
2. Wartość właściwości JMS_IBM_Character_Set jest wartością typu String, która zawiera zestaw znaków Java odpowiadający wartości liczbowej CodedCharacterSetId . Pole MQMD CodedCharacterSetId jest wartością liczbową, która zawiera odpowiednik łańcucha zestawu znaków Java określonego przez właściwość JMS_IBM_Character_Set.

Odwzorowywanie pól JMS na pola IBM MQ (komunikaty wychodzące)
Tabele te pokazują, w jaki sposób nagłówki i pola właściwości JMS są odwzorowywane na pola MQMD i MQRFH2 w czasie wysyłania () lub publikowania ().

Tabela 24 na stronie 150 pokazuje, w jaki sposób pola nagłówka JMS są odwzorowywane na pola MQMD/RFH2 w czasie wysyłania () lub publikowania (). Program Tabela 25 na stronie 151 pokazuje, w jaki sposób właściwości JMS są odwzorowywane na pola MQMD/RFH2 w czasie wysyłania () lub publikowania (). Tabela 26 na stronie 152 pokazuje, w jaki sposób właściwości specyficzne dla dostawcy JMS są odwzorowywane na pola MQMD w czasie wysyłania () lub publikowania (),

W przypadku pól oznaczonych przez obiekt Message Object nadawana wartość jest wartością przechowaną w komunikacie JMS bezpośrednio przed operacją send () lub publish (). Wartość podana w komunikacie JMS pozostaje niezmieniona przez operację.

W przypadku pól oznaczonych metodą Send Method (Set by Send Method) wartość jest przypisywany podczas wykonywania funkcji send () lub publikowania (dowolna wartość, która jest wstrzymana w komunikacie JMS , jest ignorowana). Wartość w komunikacie JMS zostanie zaktualizowana w celu wyświetlenia użytej wartości.

Pola oznaczone jako Odbierz-tylko nie są przesyłane i pozostają niezmienione w wiadomości przez funkcję send () lub publish ().

Tabela 24. Odwzorowanie pola komunikatu wychodzącego			
Nazwa pola nagłówka JMS	**Pole MQMD używane do transmisji**	**Nagłówek**	**Ustawione przez**
JMSDestination		MQRFH2	Metoda wysyłania

Tabela 24. Odwzorowanie pola komunikatu wychodzącego (kontynuacja)			
Nazwa pola nagłówka JMS	**Pole MQMD używane do transmisji**	**Nagłówek**	**Ustawione przez**
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	ReplyToQ/ReplyToQMgr	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 25. Odwzorowanie właściwości komunikatu wychodzącego JMS			
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 należy upewnić 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`.

Następujące właściwości mogą zostać przesłonięte przez aplikację:

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 26. Odwzorowanie właściwości specyficznej 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
Wygaśnięcie JMS_IBM_Report_Expiration	Raport		Obiekt komunikatu
JMS_IBM_Report_COA/COD	Raport		Obiekt komunikatu
Plik JMS_IBM_Report_NAN/PAN	Raport		Obiekt komunikatu
JMS_IBM_Report_Pass_Msg_ID	Raport		Obiekt komunikatu
JMS_IBM_Report_Pass_Correl_ID	Raport		Obiekt komunikatu
JMS_IBM_Report_Discard_Msg	Raport		Obiekt komunikatu
JMS_IBM_MsgType	MsgType		Obiekt komunikatu
Opinie JMS_IBM_Feedback	Opinie		Obiekt komunikatu
Format 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
JMS_IBM_PutTime	PutTime		Metoda wysyłania
Grupa JMS_IBM_Last_Msg_In_Group	MsgFlags		Obiekt komunikatu

Odwzorowywanie pól nagłówka produktu JMS przy użyciu funkcji send () lub publikowania ()
Te uwagi odnoszą się do odwzorowania pól JMS w funkcji send () lub publikowania ().

Miejsce JMSDestination do MQRFH2
Jest on zapisywany jako łańcuch, który przekształca do postaci szeregowej parametry salient obiektu docelowego, tak aby odbierający JMS mógł odtworzyć równoważny obiekt docelowy. Pole MQRFH2 jest zakodowane jako identyfikator URI (szczegółowe informacje na temat notacji URI zawiera sekcja "Identyfikatory URI (Uniform Resource Identifier)" na stronie 213).

JMSReplyTo do MQMD.ReplyToQ, ReplyToQMgr, MQRFH2
Nazwa kolejki jest kopiowana do MQMD.ReplyToQ , a nazwa menedżera kolejek jest kopiowana do pól ReplyToQMgr. Informacje o rozszerzeniu miejsca docelowego (inne przydatne szczegóły, które są przechowywane w obiekcie docelowym), są kopiowane do pola MQRFH2 . Pole MQRFH2 jest zakodowane jako identyfikator URI (patrz sekcja "Identyfikatory URI (Uniform Resource Identifier)" na stronie 213 , aby uzyskać szczegółowe informacje na temat notacji identyfikatora URI).

JMSDeliveryMode do MQMD.Persistence
Wartość JMSDeliveryMode jest ustawiana przez metodę send () lub publish () lub MessageProducer, chyba że obiekt docelowy przesłoni ją. Wartość JMSDeliveryMode jest odwzorowana na wartość MQMD.Persistence w następujący sposób:

- JMS wartość PERSISTENT jest równoważna wartości MQPER_PERSISTENT
- JMS wartość NON_PERSISTENT jest równoważna wartości MQPER_NOT_PERSISTENT

Jeśli właściwość trwałości MQQueue nie jest ustawiona na wartość WMQConstants.WMQ_PER_QDEF, wartość trybu dostarczania jest również zakodowana w tabeli MQRFH2.

JMSExpiration do/z MQMD.Expiry, MQRFH2
JMSExpiration przechowuje czas do utraty ważności (suma czasu bieżącego i czasu życia), podczas gdy menedżer MQMD zapisuje czas życia. Ponadto wartość JMSExpiration jest podana w milisekundach, ale MQMD.Expiry jest w dziesiątych częściach sekundy.

- Jeśli metoda send () ustawia nieograniczony czas na życie, MQMD.Expiry jest ustawiony na wartość MQEI_UNLIMITED, a wartość JMSExpiration nie jest zakodowana w tabeli MQRFH2.
- Jeśli metoda send () ustawia czas na życie, który jest mniejszy niż 214748364.7 sekund (około 7 lat), czas życia jest zapisywany w strukturze MQMD.Expiry, a czas utraty ważności (w milisekundach) jest kodowany jako wartość i8 w MQRFH2.
- Jeśli metoda send () ustawia czas życia większy niż 214748364.7 sekund, MQMD.Expiry jest ustawiony na wartość MQEI_UNLIMITED. Rzeczywisty czas utraty ważności (w milisekundach) jest kodowany jako wartość i8 w MQRFH2.

JMSPriority dla MQMD.Priority
Bezpośrednio odwzoruj wartość właściwości JMSPriority (0-9) na wartość priorytetu MQMD (0-9). Jeśli właściwość JMSPriority jest ustawiona na wartość inną niż domyślna, poziom priorytetu jest również zakodowany w tabeli MQRFH2.

JMSMessageID z MQMD.MessageID
Wszystkie komunikaty wysłane z programu JMS mają unikalne identyfikatory komunikatów przypisane przez produkt IBM MQ. Przypisana wartość jest zwracana w strukturze MQMD.MessageId po wywołaniu MQPUT i jest przekazywany z powrotem do aplikacji w polu JMSMessageID . Parametr IBM MQ messageId jest 24-bajtową wartością binarną, podczas gdy JMSMessageID jest łańcuchem. Wartość JMSMessageID składa się z binarnej wartości messageId przekształconej w sekwencję o długości 48 znaków szesnastkowych, poprzedzonej przedrostkiem ID:. Produkt JMS udostępnia wskazówkę, która może zostać ustawiona w celu wyłączenia tworzenia identyfikatorów komunikatów. Ta wskazówka jest ignorowana, a we wszystkich przypadkach przypisany jest unikalny identyfikator. Każda wartość, która jest ustawiona w polu JMSMessageID przed nadpisaniem komendy send ().

Jeśli wymagane jest określenie wartości MQMD.MessageID, można to zrobić za pomocą jednego z rozszerzeń IBM MQ JMS opisanych w "Odczytywanie i zapisywanie deskryptora komunikatu z aplikacji IBM MQ classes for JMS" na stronie 231.

JMSTimestamp do MQRFH2
Podczas wysyłania pole JMSTimestamp jest ustawiane zgodnie z zegarem maszyny JVM. Ta wartość jest ustawiana w MQRFH2. Każda wartość, która jest ustawiona w polu JMSTimestamp przed nadpisaniem () jest nadpisywana. 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 z MQMD.CorrelId, MQRFH2
Element JMSCorrelationID może zawierać jedną z następujących wartości:

Identyfikator komunikatu specyficznego dla dostawcy
Jest to identyfikator komunikatu z wcześniej wysłanego lub odebranego komunikatu, dlatego powinien on być łańcuchem o długości 48 małych cyfr szesnastkowych, poprzedzonym przedrostkiem `ID:` . Przedrostek jest usuwany, pozostałe znaki są przekształcane w plik binarny, a następnie są ustawiane w MQMD.CorrelId , pole. Wartość CorrelId nie jest zakodowana w tabeli MQRFH2.

Dostawca-wartość bajtu rodzimego []
Wartość jest kopiowana do MQMD.CorrelId -dopełnione wartościami pustymi lub obcięte do 24 bajtów, jeśli jest to konieczne. Wartość CorrelId nie jest zakodowana w tabeli MQRFH2.

Łańcuch specyficzny dla aplikacji
Wartość zostanie skopiowana do pliku MQRFH2. Pierwsze 24 bajty łańcucha, w formacie UTF8 , są zapisywane w MQMD.CorrelID.

Odwzorowanie pól właściwości JMS
Te uwagi odnoszą się do odwzorowania pól właściwości produktu JMS w komunikatach produktu IBM MQ .

JMSXUserID z tabeli MQMD UserIdentifier
Wartość JMSXUserID jest ustawiona w przypadku powrotu z wywołania wysyłania.

JMSXAppID z nazwy MQMD PutAppl
JSMXAppID jest ustawiona w przypadku powrotu z wywołania wysyłania.

JMSXGroupID do MQRFH2 (punkt-punkt)
W przypadku komunikatów typu punkt z punktem identyfikator JMSXGroupID jest kopiowany do pola GroupID deskryptora MQMD. Jeśli identyfikator JMSXGroupID zostanie uruchomiony z przedrostkiem ID:, zostanie on przekształcony w plik binarny. W przeciwnym razie jest on zakodowany jako łańcuch UTF8 . Wartość jest dopełniona lub obcięta, jeśli jest to konieczne, do długości 24 bajtów. Flaga MQMF_MSG_IN_GROUP jest ustawiona.

JMSXGroupID do MQRFH2 (publish/subscribe)
W przypadku komunikatów publikowania/subskrypcji wartość JMSXGroupID jest kopiowana do łańcucha MQRFH2 jako łańcuch.

JMSXGroupSeq MQMD MsgSeqLiczba (punkt-do-punktu)
W przypadku komunikatów w trybie punkt z punktem JMSXGroupSeq jest kopiowany do pola MsgSeqNumber w tabeli MQMD. Flaga MQMF_MSG_IN_GROUP jest ustawiona.

JMSXGroupSeq MQMD MsgSeqLiczba (publish/subscribe)
W przypadku komunikatów publikowania/subskrypcji wartość JMSXGroupSeq jest kopiowana do pliku 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.

Odwzorowywanie pól IBM MQ na pola JMS (komunikaty przychodzące)
Tabele te pokazują, w jaki sposób pola nagłówka i właściwości produktu JMS są odwzorowywane na pola MQMD i MQRFH2 w czasie get () lub receive ().

Tabela 27 na stronie 156 pokazuje, w jaki sposób pola nagłówka JMS są odwzorowywane na pola MQMD/MQRFH2 w czasie get () lub receive (). Program Tabela 28 na stronie 157 pokazuje, w jaki sposób pola właściwości JMS są odwzorowywane na pola MQMD/MQRFH2 w czasie get () lub receive (). Tabela 29 na stronie 157 pokazuje, w jaki sposób odwzorowywane są właściwości specyficzne dla dostawcy JMS .

Tabela 27. Odwzorowanie pola nagłówka JMS komunikatu przychodzącego

Nazwa pola nagłówka JMS	**Pobrano pole MQMD z**	**Pole MQRFH2 pobrane z**
JMSDestination		jms.Dst lub mqps.Top "1" na stronie 156
JMSDeliveryMode	Trwałość "2" na stronie 157	jms.Dlv "2" na stronie 157
JMSExpiration		jms.Exp
JMSPriority	Priorytet	
JMSMessageID	MsgID	
JMSTimestamp	PutDate "2" na stronie 157 PutTime "2" na stronie 157	jms.Tms "2" na stronie 157
JMSCorrelationID	CorrelId "2" na stronie 157	jms.Cid "2" na stronie 157
JMSReplyTo	Kolejka_zwrotna "2" na stronie 157 Menedżer_kolejek_zwrotnych "2" na stronie 157	jms.Rto "2" na stronie 157
JMSType		mcd.Type, mcd.Set, mcd.Fmt
JMSRedelivered	BackoutCount	

Uwaga:

1. Jeśli ustawiona jest wartość zarówno jms.Dst , jak i mqps.Top , używana jest wartość w jms.Dst .

2. W przypadku właściwości, które mogą mieć wartości pobrane z MQRFH2 lub MQMD, jeśli oba te wartości są dostępne, używane jest ustawienie w tabeli MQRFH2 .
3. Wartość właściwości JMS_IBM_Character_Set jest wartością typu String, która zawiera zestaw znaków Java odpowiadający wartości liczbowej CodedCharacterSetId .

Tabela 28. Odwzorowanie właściwości komunikatu przychodzącego

JMS Nazwa właściwości	**Pobrano pole MQMD z**	**Pole MQRFH2 pobrane z**
JMSXUserID	UserIdentifier	
JMSXAppID	Nazwa_aplikacji_wstawiającej	
JMSXDeliveryCount	BackoutCount	
JMSXGroupID	GroupId "1" na stronie 157	jms.Gid "1" na stronie 157
JMSXGroupSeq	Numer_kolejny_komunikatu"1" na stronie 157	jms.Seq "1" na stronie 157

Uwaga:

1. W przypadku właściwości, które mogą mieć wartości pobrane z MQRFH2 lub MQMD, jeśli oba te wartości są dostępne, używane jest ustawienie w tabeli 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 29. Odwzorowanie właściwości JMS specyficzne dla dostawcy komunikatów przychodzących

JMS Nazwa właściwości	**Pobrano pole MQMD z**	**Pole MQRFH2 pobrane z**
Wyjątek JMS_IBM_Report_Exception	Raport	
Wygaśnięcie JMS_IBM_Report_Expiration	Raport	
Plik JMS_IBM_Report_COA	Raport	
JMS_IBM_Report_COD	Raport	
Plik JMS_IBM_Report_PAN	Raport	
Plik JMS_IBM_Report_NAN	Raport	
JMS_IBM_Report_ Pass_Msg_ID	Raport	
JMS_IBM_Report_Pass_Correl_ID	Raport	
JMS_IBM_Report_Discard_Msg	Raport	
JMS_IBM_MsgType	MsgType	
Opinie JMS_IBM_Feedback	Opinie	
Format JMS_IBM_Format	Format	
Typ JMS_IBM_PutAppl	Typ_aplikacji_wstawiającej	
JMS_IBM_Encoding "1" na stronie 158	Kodowanie	
JMS_IBM_Character_Set "1" na stronie 158	CodedCharacterSetId	
JMS_IBM_PutDate	PutDate	
JMS_IBM_PutTime	PutTime	

Tabela 29. Odwzorowanie właściwości JMS specyficzne dla dostawcy komunikatów przychodzących (kontynuacja)		
JMS Nazwa właściwości	**Pobrano pole MQMD z**	**Pole MQRFH2 pobrane z**
Grupa JMS_IBM_Last_Msg_In_Group	MsgFlags	

1. Ustaw tylko wtedy, gdy komunikat przychodzący jest komunikatem bajtów.

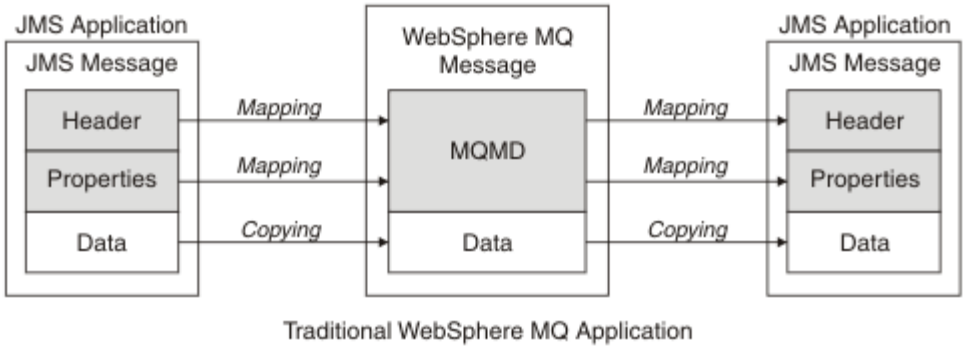
Wymiana komunikatów między aplikacją JMS a tradycyjną aplikacją IBM MQ
W tej sekcji opisano, co się dzieje, gdy aplikacja JMS wymienia komunikaty z tradycyjną aplikacją IBM MQ , która nie może przetworzyć nagłówka MQRFH2 .

Rysunek 11 na stronie 158 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 tworzony żaden nagłówek MQRFH2 . Jeśli ta opcja nie zostanie wykonana, aplikacja odbierający musi mieć możliwość obsługi nagłówka MQRFH2 .

Odwzorowanie z produktu JMS na deskryptor MQMD skierowane do tradycyjnej aplikacji produktu IBM MQ jest takie samo, jak odwzorowanie z produktu JMS na deskryptor MQMD docelowy w aplikacji JMS . Jeśli program IBM MQ classes for JMS otrzyma komunikat IBM MQ z polem MQMD `Format` ustawionym na wartość inną niż MQFMT_RFH2, dane są odbierane z aplikacji innej niżJMS . Jeśli formatem jest MQFMT_STRING, komunikat jest odbierany jako komunikat tekstowy produktu JMS . W przeciwnym razie zostanie odebrany jako komunikat w postaci bajtów JMS . Ponieważ nie ma komendy MQRFH2, można odtworzyć tylko te właściwości produktu JMS , które są przesyłane w strukturze MQMD.

Jeśli program IBM MQ classes for JMS odbierze komunikat, który nie ma nagłówka MQRFH2 , właściwość TARGCLIENT obiektu kolejki lub tematu pochodzącego z pola nagłówka JMSReplyTo komunikatu jest domyślnie ustawiona na wartość MQ . Oznacza to, że komunikat odpowiedzi wysłany do kolejki lub tematu również nie ma nagłówka MQRFH2 . Można wyłączyć to zachowanie, w tym nagłówek MQRFH2 w komunikacie odpowiedzi tylko wtedy, gdy oryginalny komunikat ma nagłówek MQRFH2 , ustawiając właściwość TARGCLIENTMATCHING dla fabryki połączeń na wartość NO.



Rysunek 11. Sposób transformowania komunikatów produktu JMS na komunikaty produktu IBM MQ bez nagłówka MQRFH2 .

Treść komunikatu JMS
Ten temat zawiera informacje na temat kodowania samego treści komunikatu. Kodowanie zależy od typu komunikatu produktu JMS .

ObjectMessage
Obiekt ObjectMessage jest obiektem serializowanym przez komponent Java Runtime w normalny sposób.

TextMessage
Obiekt TextMessage jest zakodowanym łańcuchem. W przypadku komunikatu wychodzącego łańcuch jest zakodowany w zestawie znaków określonym przez obiekt docelowy. Wartością domyślną jest

kodowanie UTF8 (kodowanie UTF8 rozpoczyna się od pierwszego znaku komunikatu; na początku pola nie ma pola długości). Możliwe jest jednak określenie dowolnego innego zestawu znaków obsługiwanego przez produkt IBM MQ classes for JMS. Takie zestawy znaków są używane głównie w przypadku wysyłania komunikatu do aplikacji innej niżJMS .

Jeśli zestaw znaków jest zestawem typu double-byte (w tym UTF16), to specyfikacja kodowania liczb całkowitych obiektu docelowego określa kolejność bajtów.

Komunikat przychodzący jest interpretowany przy użyciu zestawu znaków i kodowania, które są określone w samym komunikacie. Te specyfikacje znajdują się w ostatnim nagłówku produktu IBM MQ (lub w nagłówku MQMD, jeśli nie ma nagłówków). W przypadku komunikatów produktu JMS ostatni nagłówek jest zwykle nagłówkiem MQRFH2.

BytesMessage

BytesMessage jest domyślnie sekwencją bajtów zdefiniowaną w specyfikacji JMS 1.0.2 i powiązanej z nią dokumentacji produktu Java .

W przypadku komunikatu wychodzącego, który został złożony przez samą aplikację, właściwość kodowania obiektu docelowego może zostać użyta do przesłonięcia kodowania pól liczb całkowitych i zmiennopozycyjnych zawartych w komunikacie. Na przykład można zażądać, aby wartości zmiennopozycyjne były przechowywane w S/390 , a nie w formacie IEEE).

Komunikat przychodzący jest interpretowany przy użyciu kodowania liczbowego określonego w samym komunikacie. Ta specyfikacja znajduje się w ostatnim nagłówku produktu IBM MQ (lub w nagłówku MQMD, jeśli nie ma nagłówków). W przypadku komunikatów produktu JMS ostatni nagłówek jest zwykle nagłówkiem MQRFH2.

Jeśli zostanie odebrana wartość BytesMessage i zostanie ona ponownie wysłana bez modyfikacji, jej treść jest przesyłana bajtem za bajt, ponieważ została odebrana. Właściwość kodowania obiektu docelowego nie ma wpływu na treść. Jedynym obiektem typu łańcuchowego, który może być jawnie wysyłany w BytesMessage jest łańcuch UTF8 . Jest on zakodowany w formacie Java UTF8 i rozpoczyna się od pola o długości 2 bajtów. Właściwość zestawu znaków dla obiektu docelowego nie ma wpływu na kodowanie wychodzącego elementu BytesMessage. Wartość zestawu znaków w przychodzącym komunikacie IBM MQ nie ma wpływu na interpretację tego komunikatu jako JMS BytesMessage(BytesMessage).

Mało prawdopodobne jest, aby aplikacje inne niżJava rozpoznały kodowanie Java UTF8 . Dlatego w przypadku aplikacji produktu JMS w celu wysłania komunikatu BytesMessage zawierającego dane tekstowe sama aplikacja musi przekształcić swoje łańcuchy w tablice bajtów i zapisać te tablice bajtowe w polu BytesMessage.

MapMessage

Element MapMessage jest łańcuchem zawierającym nazwy XML/typ/wartości 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 20 na stronie 147. Domyślnym typem danych jest `string`, a więc atrybut `dt="string"` jest pomijany w przypadku elementów łańcuchowych.

Zestaw znaków używany do kodowania lub interpretowania łańcucha XML, który tworzy treść komunikatu mapy, jest określany zgodnie z regułami, które mają zastosowanie do komunikatu tekstowego.

Wersje produktu IBM MQ classes for JMS w wersji wcześniejszej niż 5.3 zakodowane w treści komunikatu mapy w następującym formacie:

```
<map>
  <elementname1 dt="datatype1">value1</elementname1>
  <elementname2 dt="datatype2">value2</elementname2>
```

```
  ...
</map>
```

5.3 i nowsze wersje produktu IBM MQ classes for JMS mogą interpretować albo format, ale wersje produktu IBM MQ classes for JMS wcześniejsze niż 5.3 nie mogą interpretować bieżącego formatu.

Jeśli aplikacja musi wysłać komunikaty odwzorowania do innej aplikacji, która korzysta z wersji produktu IBM MQ classes for JMS wcześniejszej niż 5.3, aplikacja wysyłający musi wywołać metodę fabryki połączeń `setMapNameStyle(WMQConstants.WMQ_MAP_NAME_STYLE_COMPATIBLE)` , aby określić, że komunikaty mapy są wysyłane w poprzednim formacie. Domyślnie wszystkie komunikaty mapy są wysyłane w bieżącym formacie.

StreamMessage
Komunikat StreamMessage jest podobny do komunikatu mapy, 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 20 na stronie 147. Domyślnym typem danych jest `string`, a więc atrybut `dt="string"` jest pomijany w przypadku elementów łańcuchowych.

Zestaw znaków używany do kodowania lub interpretowania łańcucha XML, który składa się z treści StreamMessage , jest określany na podstawie reguł, które mają zastosowanie do komunikatu TextMessage.

Pole MQRFH2.format jest ustawione w następujący sposób:

MQFMT_NONE
dla ObjectMessage, BytesMessagelub komunikatów bez treści.

MQFMT_STRING
dla TextMessage, StreamMessagelub MapMessage.

Konwersja komunikatów JMS
Konwersja danych komunikatu w produkcie JMS jest wykonywana podczas wysyłania i odbierania komunikatów. Program IBM MQ automatycznie wykonuje większość konwersji danych. Konwertuje dane tekstowe i liczbowe podczas przesyłania komunikatów między aplikacjami produktu 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 nietekstowych między aplikacją IBM MQ a aplikacją JMS .
- Wymiana danych tekstowych w formacie bajtowym.
- Przekształcanie tekstu w aplikacji.

JMS Dane komunikatów

Konwersja danych jest konieczna do wymiany danych tekstowych i liczbowych między aplikacjami, nawet między dwoma aplikacjami produktu JMS . Wewnętrzna reprezentacja tekstu i liczb musi być zakodowana w taki sposób, aby mogły zostać przesłane w komunikacie. Kodowanie wymusza podjęcie decyzji o tym, w jaki sposób reprezentowane są liczby i tekst. Produkt IBM MQ zarządza kodowaniem tekstu i liczb w komunikatach JMS , z wyjątkiem `JMSObjectMessage`, patrz “JMSObjectMessage” na stronie 167. Korzysta ona z trzech atrybutów komunikatu. Trzy atrybuty to: `CodedCharacterSetId`, `Encoding`i `Format`.

Te trzy atrybuty komunikatu są zwykle zapisywane w nagłówku JMS , `MQRFH2`, w polach komunikatu JMS . Jeśli typ komunikatu to `MQ`, a nie typ komunikatu `JMS` , atrybuty są zapisywane w deskryptorze

komunikatu MQMD. Atrybuty są używane do przekształcania danych komunikatu produktu JMS . Dane komunikatu programu JMS są przesyłane w części danych komunikatu w komunikacie IBM MQ .

JMS Właściwości komunikatu

Właściwości komunikatu produktu JMS , takie jak JMS_IBM_CHARACTER_SET, są wymieniane w części nagłówka MQRFH2 komunikatu produktu JMS , chyba że komunikat został wysłany bez MQRFH2. Tylko produkty JMSTextMessage i JMSBytesMessage mogą być wysyłane bez MQRFH2. Jeśli właściwość JMS jest przechowywana jako właściwość komunikatu IBM MQ w deskryptorze komunikatu, MQMD, jest ona przekształcana w ramach konwersji MQMD . Jeśli właściwość JMS jest przechowywana w MQRFH2, jest ona przechowywana w zestawie znaków określonym przez produkt MQRFH2.NameValueCCSID. Po wysłaniu lub odebraniu komunikatu właściwości komunikatu są przekształcane do i z ich wewnętrznej reprezentacji w maszynie JVM. Konwersja jest ustawiona na i z zestawu znaków deskryptora komunikatu lub MQRFH2.NameValueCCSID. Dane liczbowe są przekształcane w tekst.

Konwersja komunikatów JMS

Poniższe tematy zawierają przykłady i zadania, które są przydatne, jeśli planowane jest wymianę 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 161
2. “Konwersja danych aplikacji” na stronie 162
3. “Konwersja danych menedżera kolejek” na stronie 163
4. “Konwersja danych kanału komunikatów” na stronie 163

Gdzie należy przeprowadzić konwersję?
W sekcji “Wybór podejścia do konwersji komunikatów: odbiorca jest dobry” na stronie 164opisano 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 30 na stronie 165.

[1] "JMS Klient" oznacza IBM MQ classes for JMS implementujące interfejs JMS , który działa w trybie klienta lub w trybie powiązań.

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 174.

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 166 i Rysunek 15 na stronie 166. 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` `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`, `JMSMapMessage`lub `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.

[2] Jeden wyjątek: dane zapisane przy użyciu parametru `writeUTF` rozpoczynają się od pola o długości 2 bajtów

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 przy użyciu aplikacji innej niżJMS” na stronie 182.

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 166). 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 163.

```
((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 (`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 przy użyciu aplikacji innej niżJMS” na stronie 182.

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 UNIX. `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 przy użyciu aplikacji innej niżJMS” na stronie 182. Przykłady konwersji komunikatów wymienianych z aplikacjami innymi niżJMS .

Przykłady

Tabela typów komunikatów i typów konwersji

Tabela 30. Typy komunikatów i typy konwersji

	Typ konwersji			
Typ komunikatu	**Tekstowy**	**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
JMSStreamMessage	readString writeString	readDouble readFloat readInt readLong readShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readBytes readChar writeByte writeBytes writeChar
JMSMapMessage	getString setString	getDouble getFloat getInt getLong getShort setDouble setFloat setInt setLong setShort	getBoolean getObject setBoolean setObject	getByte getBytes readChar setByte setBytes setChar

```
gmo.Options = MQGMO_WAIT         /* wait for new messages           */
            | MQGMO_NO_SYNCPOINT /* no transaction                  */
            | MQGMO_CONVERT;     /* convert if necessary            */

     while (CompCode != MQCC_FAILED) {
     buflen = sizeof(buffer) - 1; /* buffer size available for GET   */
     memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
     memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
     md.Encoding       = MQENC_NATIVE;
     md.CodedCharSetId = MQCCSI_Q_MGR;

           MQGET(Hcon,          /* connection handle               */
           Hobj,                /* object handle                   */
           &md,                 /* message descriptor              */
           &gmo,                /* get message options             */
           buflen,              /* buffer length                   */
           buffer,              /* message buffer                  */
           &messlen,            /* message length                  */
           &CompCode,           /* completion code                 */
           &Reason);            /* reason code                     */
```

Rysunek 13. Fragment kodu z pliku amqsget0.c

Wysyłanie i odbieranie tekstu w JMSBytesMessage

Kod w pliku Rysunek 14 na stronie 166 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 166. 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
Na liście znajdują się metody używane do konwersji i kodowania komunikatów klienta JMS , z przykładami kodu 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 przy użyciu aplikacji innej niżJMS
Wykonaj kroki proponowane w tym zadaniu, aby zaprojektować i zbudować wyjście konwersji danych, a także aplikację kliencką JMS , która może wymieniać komunikaty z aplikacją inną niżJMS przy użyciu produktu JMSBytesMessage. Wymiana sformatowanego komunikatu z aplikacją inną niżJMS może odbywać się bez wywoływania wyjścia konwersji danych lub bez niej.

Odsyłacze pokrewne

Typy komunikatów i konwersje produktu JMS
Wybór typu komunikatu wpływa na podejście do konwersji komunikatów. Interakcja konwersji komunikatów i typu komunikatu jest opisana dla typów komunikatów JMS , JMSObjectMessage, JMSTextMessage, JMSMapMessage, JMSStreamMessagei JMSBytesMessage.

Typy komunikatów i konwersje produktu JMS
Wybór typu komunikatu wpływa na podejście do konwersji komunikatów. Interakcja konwersji komunikatów i typu komunikatu jest opisana dla typów komunikatów JMS , JMSObjectMessage, JMSTextMessage, JMSMapMessage, JMSStreamMessagei JMSBytesMessage.

JMSObjectMessage

JMSObjectMessage zawiera jeden obiekt i wszystkie obiekty, do których się odwołuje, przekształcony do postaci szeregowej strumienia bajtów przez maszynę JVM. Tekst jest przekształcany do postaci szeregowej w produkcie UTF-8i jest ograniczony do łańcuchów lub tablic znakowych o długości nie większej niż 65534 bajtów. Zaletą produktu JMSObjectMessage jest to, że aplikacje nie są zaangażowane w żadne problemy związane z konwersją danych, o ile tylko te metody i atrybuty są używane tylko w tych metodach. Produkt JMSObjectMessage udostępnia konwersję danych dla obiektów złożonych bez programisty aplikacji rozważając, jak zakodować obiekt w komunikacie. Wadą korzystania z produktu JMSObjectMessage jest możliwość wymiany tylko z innymi aplikacjami produktu JMS . Wybierając jeden z pozostałych typów komunikatów produktu JMS , możliwe jest wymianę komunikatów produktu JMS z aplikacjami innymi niżJMS .

“Wysyłanie i odbieranie JMSObjectMessage” na stronie 170 przedstawia obiekt String , który jest wymieniany w komunikacie.

Aplikacja kliencka JMS może odbierać JMSObjectMessage tylko w komunikacie, który ma treść w stylu JMS. Miejsce docelowe musi określać treść stylu JMS .

JMSTextMessage

JMSTextMessage zawiera pojedynczy łańcuch tekstowy. Gdy wysyłany jest komunikat tekstowy, tekst Format jest ustawiany na wartość "MQSTR ", WMQConstants.MQFMT_STRING. CodedCharacterSetId tekstu jest ustawiany na identyfikator kodowanego zestawu znaków zdefiniowany dla miejsca docelowego. Tekst jest zakodowany w CodedCharacterSetId przez IBM MQ. Pola CodedCharacterSetId i Format są ustawione w deskryptorze komunikatu, MQMDlub w polach JMS w MQRFH2. Jeśli komunikat jest zdefiniowany jako mający styl treści komunikatu produktu WMQ_MESSAGE_BODY_MQ lub styl treści nie został określony, ale miejscem docelowym jest WMQ_TARGET_DEST_MQ, to pola deskryptora komunikatu są ustawiane. W przeciwnym razie komunikat ma JMS RFH2 , a pola są ustawiane w stałej części MQRFH2.

Aplikacja może przesłonić identyfikator kodowanego zestawu znaków zdefiniowany dla miejsca docelowego. Musi ona ustawić właściwość komunikatu JMS_IBM_CHARACTER_SET na identyfikator kodowanego zestawu znaków; patrz przykład w sekcji “Wysyłanie i odbieranie JMSTextmessage” na stronie 170.

Gdy klient JMS wywoła 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. Menedżer kolejek przekształca komunikat tekstowy z JMS_IBM_CHARACTER_SET określonego dla komunikatu przed przesłaniem komunikatu do klienta JMS . Zestaw znaków przekształconego komunikatu to 1208, UTF-8, chyba że miejsce docelowe ma inną wartość WMQ_RECEIVE_CCSID. CodedCharacterSetId

w komunikacie, który odwołuje się do JMSTextMessage , jest aktualizowany do docelowego identyfikatora zestawu znaków. Tekst jest zdekodowany z docelowego zestawu znaków w kodzie Unicode za pomocą metody getText . Patrz przykład w sekcji “Wysyłanie i odbieranie JMSTextmessage” na stronie 170.

JMSTextMessage może być wysyłany w treści komunikatu w stylu MQbez nagłówka JMS MQRFH2 . Wartość atrybutów miejsca docelowego, WMQ_MESSAGE_BODY i WMQ_TARGET_DEST określa styl treści komunikatu, o ile nie zostanie przesłonięty przez aplikację.
Aplikacja może przesłonić wartości ustawione w miejscu docelowym przez wywołanie funkcji destination.setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ) lub destination.setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ).

Jeśli produkt JMSTextMessage jest wysyłany z treścią stylu MQ , wysyłając je do miejsca docelowego o nazwie WMQ_MESSAGE_BODY ustawionej na wartość WMQ_MESSAGE_BODY_MQ, nie można odbierać go 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. W przypadku próby odebrania komunikatu jako JMSTextMessage powoduje to wyjątek ClassCastException: com.ibm.jms.JMSBytesMessage cannot be cast to javax.jms.TextMessage.

Uwaga: Tekst w JMSBytesMessage nie jest przekształcany przez klienta JMS. Klient może odbierać tekst tylko w komunikacie jako tablicę bajtów. Jeśli konwersja menedżera kolejek jest włączona, tekst jest przekształcany przez menedżer kolejek, ale klient JMS musi nadal odbierać go jako tablicę bajtów w JMSBytesMessage.

Zwykle lepszym zastosowaniem jest użycie właściwości WMQ_TARGET_DEST w celu określenia, czy produkt JMSTextMessage jest wysyłany z użyciem stylu treści MQ czy JMS. Następnie można odebrać komunikat z miejsca docelowego, które ma wartość WMQ_TARGET_DEST ustawioną na 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 produktu JMS są podobne. Typy podstawowe można odczytywać i zapisywać na komunikatach przy użyciu metod opartych na interfejsach DataInputStream i DataOutputStream . Patrz sekcja “Tabela typów komunikatów i typów konwersji” na stronie 172. Szczegółowe informacje są opisane w sekcji “Konwersja i kodowanie komunikatów klienta JMS” na stronie 174. Każda operacja podstawowa jest oznaczona; patrz “Treść komunikatu JMS” na stronie 158.

Dane liczbowe są odczytywaniei zapisywane w komunikacie zakodowanym jako tekst XML. Do właściwości miejsca docelowego nie jest odwołanie JMS_IBM_ENCODING. Dane tekstowe są traktowane w taki sam sposób, jak tekst w JMSTextMessage. Jeśli użytkownik miał przejrzeć treść komunikatu utworzoną na podstawie przykładu w programie Rysunek 20 na stronie 171, wszystkie dane komunikatu będą w kodzie EBCDIC, ponieważ zostały wysłane z wartością zestawu znaków 37.

Istnieje możliwość wysłania wielu 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 do typu, który został wysłany, typ jest przekształcany. Jeśli typ nie może zostać przekształcony, zgłaszany jest wyjątek. Szczegółowe informacje na ten temat zawiera sekcja Klasa JMSStreamMessage . Przykład w sekcji “Wysyłanie danych w serwerach JMSStreamMessage i JMSMapMessage” na stronie 171 ilustruje konwersję typu i pobieranie treści JMSMapMessage z sekwencji.

Pole MQRFH2.format dla partycji JMSMapMessage i JMSStreamMessage jest ustawione na wartość "MQSTR ". Jeśli właściwość miejsca docelowego WMQ_RECEIVE_CONVERSION jest ustawiona na 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. MQRFH2.Encoding to

Native. If WMQ_RECEIVE_CONVERSION is WMQ_RECEIVE_CONVERSION_CLIENT_MSG the CodedCharacterSetId and Encoding of the MQRFH2 is the value set by the sender.

Aplikacja kliencka JMS może odbierać JMSMapMessage lub JMSStreamMessage tylko w komunikacie, który ma treść w stylu JMS, oraz z miejsca docelowego, które nie określa treści w stylu MQ .

JMSBytesMessage

JMSBytesMessage może zawierać wiele typów podstawowych. Typy podstawowe można odczytywać i zapisywać na komunikatach przy użyciu metod opartych na interfejsach DataInputStream i DataOutputStream . Patrz sekcja “Tabela typów komunikatów i typów konwersji” na stronie 172. Szczegółowe informacje są opisane w sekcji “Typy komunikatów i konwersje produktu JMS” na stronie 167.

Kodowanie danych liczbowych w komunikacie jest sterowane przez wartość JMS_IBM_ENCODING , która jest ustawiona przed zapisaniem danych liczbowych w JMSBytesMessage. Aplikacja może przesłonić domyślne kodowanie produktu Native zdefiniowane dla produktu JMSBytesMessage , ustawiając właściwość komunikatu JMS_IBM_ENCODING.

Dane tekstowe można odczytywać i zapisywać w programie UTF-8 przy użyciu produktów readUTF i writeUTF, a także w kodzie Unicode za pomocą metod readChar i writeChar . Nie ma metod, które używają produktu CodedCharacterSetId. Alternatywnie klient JMS może kodować i dekodować tekst w bajtach przy użyciu klasy Charset . Przesyła on bajty między maszyną JVM i komunikatem bez wykonywania konwersji przez produkt IBM MQ classes for JMS . Patrz sekcja “Wysyłanie i odbieranie tekstu w JMSBytesMessage” na stronie 171.

Produkt JMSBytesMessage wysyłany do aplikacji MQ jest zwykle wysyłany w treści komunikatu w stylu MQbez nagłówka JMS MQRFH2 . Jeśli jest ona wysyłana do aplikacji JMS , styl treści komunikatu to zwykle JMS. Wartość atrybutów miejsca docelowego, WMQ_MESSAGE_BODY i WMQ_TARGET_DEST określa styl treści komunikatu, o ile nie zostanie przesłonięty przez aplikację. Aplikacja może przesłonić wartości ustawione w miejscu docelowym przez wywołanie funkcji destination.setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ) lub destination.setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ).

Po wysłaniu JMSBytesMessage z treścią stylu MQ można odebrać komunikat z miejsca docelowego, który definiuje styl treści komunikatu w produkcie MQ lub JMS . W przypadku wysyłania JMSBytesMessage z treścią stylu JMS należy odebrać komunikat z miejsca docelowego, w którym zdefiniowano styl treści komunikatu produktu JMS . W przeciwnym razie produkt MQRFH2 będzie traktowany jako część danych komunikatu użytkownika, co może nie być tym, czego się spodziewa.

Niezależnie od tego, czy komunikat ma styl treści w programie MQ , czy w produkcie JMS , ustawienie WMQ_TARGET_DESTnie ma wpływu na sposób jego odebrania.

Komunikat może zostać później przetransformowany przez menedżer kolejek, jeśli dla danych komunikatu podano Format , a konwersja danych menedżera kolejek jest włączona. Nie należy używać pola formatu dla niczego innego niż określenie formatu danych komunikatu lub pozostawienie pustego pola, MQConstants.MQFMT_NONE

Istnieje możliwość wysłania wielu elementów w JMSBytesMessage. Każda pozycja liczbowa jest przekształcana po wysłaniu komunikatu z użyciem kodowania zdefiniowanego dla komunikatu.

Istnieje możliwość pobrania poszczególnych elementów danych z programu JMSBytesMessage. Wywołaj metody odczytu w tej samej kolejności, w której wywołano metody zapisu w celu utworzenia komunikatu. Każdy element numeryczny jest przekształcany, gdy komunikat jest wywoływany przy użyciu wartości Encoding zapisanej w komunikacie.

W przeciwieństwie do produktów JMSMapMessage i JMSStreamMessage, produkt 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 serwerach JMSMapMessage i JMSStreamMessage. Z tego powodu należy użyć programu JMSBytesMessage do przesyłania komunikatów sformatowanych dla innych aplikacji.

Konwersja między JMSBytesMessage a DataInputStream i DataOutputStream jest przydatna w niektórych aplikacjach. Kod oparty na przykładzie, "Odczytywanie i zapisywanie komunikatów przy użyciu produktów DataInputStream i DataOutputStream" na stronie 171, jest niezbędny do korzystania z 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. W przykładzie przedstawiono tekst w różnych zestawach znaków, które są wysyłane 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 wiadomość tekstową 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 wiadomość tekstową

Wysyłanie danych w serwerach 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 serwerach JMSStreamMessage i JMSMapMessage

Wysyłanie i odbieranie tekstu w JMSBytesMessage

Kod w pliku Rysunek 21 na stronie 171 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 171. 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 programie Rysunek 23 na stronie 172 tworzy JMSBytesMessage przy użyciu 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, która ustawia właściwość JMS_IBM_ENCODING , jest przekształcana w komentarz. Instrukcja jest poprawna, jeśli zapis bezpośrednio do pliku JMSBytesMessage, ale nie ma wpływu na zapis do DataOutputStream. Liczby zapisywane w pliku DataOutputStream są kodowane w kodowaniu produktu Native . Ustawienie JMS_IBM_ENCODING nie ma żadnego efektu.

Kod w programie Rysunek 24 na stronie 172 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. Otrzymywanie JMSBytesMessage przy użyciu DataInputStream

Strona kodowa jest drukowana za pomocą właściwości strony kodowej danych komunikatu wejściowego, JMS_IBM_CHARACTER_SET. Na wejściu JMS_IBM_CHARACTER_SET jest to strona kodowa Java , a nie liczbowy identyfikator kodowanego zestawu znaków.

Tabela typów komunikatów i typów konwersji

Tabela 31. Typy komunikatów i typy konwersji

	Typ konwersji			
Typ komunikatu	**Tekstowy**	**Liczbowy**	**Inne**	**Brak**
JMSObjectMessage				getObject setObject
JMSTextMessage	getText setText			

Tabela 31. Typy komunikatów i typy konwersji (kontynuacja)				
	Typ konwersji			
Typ komunikatu	**Tekstowy**	**Liczbowy**	**Inne**	**Brak**
JMSBytesMessage	readUTF writeUTF	readDouble readFloat readInt readLong readShort readUnsignedShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readUnsignedByte readBytes readChar writeByte writeBytes writeChar
JMSStreamMessage	readString writeString	readDouble readFloat readInt readLong readShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readBytes readChar writeByte writeBytes writeChar
JMSMapMessage	getString setString	getDouble getFloat getInt getLong getShort setDouble setFloat setInt setLong setShort	getBoolean getObject setBoolean setObject	getByte getBytes readChar setByte setBytes setChar

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
Na liście znajdują się metody używane do konwersji i kodowania komunikatów klienta JMS , z przykładami kodu 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 przy użyciu aplikacji innej niżJMS
Wykonaj kroki proponowane w tym zadaniu, aby zaprojektować i zbudować wyjście konwersji danych, a także aplikację kliencką JMS , która może wymieniać komunikaty z aplikacją inną niżJMS przy użyciu produktu JMSBytesMessage. Wymiana sformatowanego komunikatu z aplikacją inną niżJMS może odbywać się bez wywoływania wyjścia konwersji danych lub bez niej.

Konwersja i kodowanie komunikatów klienta JMS
Na liście znajdują się metody używane do konwersji i kodowania komunikatów klienta JMS , z przykładami kodu każdego typu konwersji.

Konwersja i kodowanie występują, gdy operacje podstawowe lub obiekty produktu Java są odczytywanielub zapisywane do i z komunikatów produktu JMS . Konwersja jest nazywana konwersją danych klienta JMS w celu odróżnienia jej od konwersji danych menedżera kolejek i konwersji danych aplikacji. Konwersja odbywa się ściśle w przypadku, gdy dane są odczytyane z komunikatu JMS lub zapisywane w nim. Tekst jest przekształcany w wewnętrzną reprezentację 16-bitową Unicode i z wewnętrznej reprezentacji Unicode.[3]do zestawu znaków używanego dla tekstu w komunikatach. Numeric data is converted to and Java primitive numeric types to the encoding defined for the message. Niezależnie od tego, czy konwersja jest wykonywana, oraz jaki typ konwersji jest wykonywany, zależy od typu komunikatu JMS oraz operacji odczytu lub zapisu.

Tabela 32 na stronie 174 kategoryzuje metody odczytu i zapisu dla różnych typów komunikatów produktu JMS według typu wykonywanego konwersji. Typy konwersji są opisane w tekście po tabeli.

Tabela 32. Typy komunikatów i typy konwersji

	Typ konwersji			
Typ komunikatu	**Tekstowy**	**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

[3] Niektóre reprezentacje Unicode wymagają więcej niż 16 bitów. Więcej informacji zawiera sekcja Java SE.

Tabela 32. Typy komunikatów i typy konwersji (kontynuacja)				
	Typ konwersji			
Typ komunikatu	**Tekstowy**	**Liczbowy**	**Inne**	**Brak**
JMSStreamMessage	readString writeString	readDouble readFloat readInt readLong readShort writeDouble writeFloat writeInt writeLong writeShort	readBoolean readObject writeBoolean writeObject	readByte readBytes readChar writeByte writeBytes writeChar
JMSMapMessage	getString setString	getDouble getFloat getInt getLong getShort setDouble setFloat setInt setLong setShort	getBoolean getObject setBoolean setObject	getByte getBytes readChar setByte setBytes setChar

Tekstowy

Domyślną wartością CodedCharacterSetId dla miejsca docelowego jest 1208, UTF-8. Domyślnie tekst jest przekształcany z formatu Unicode i wysyłany jako łańcuch tekstowy produktu UTF-8 . Po odebraniu tekst jest przekształcany z zakodowanego zestawu znaków w komunikacie odebranym przez klienta do kodu Unicode.

Metody setText i writeString przekształcają tekst z kodu Unicode w zestaw znaków zdefiniowany dla miejsca docelowego. Aplikacja może przesłonić docelowy zestaw znaków, ustawiając właściwość komunikatu JMS_IBM_CHARACTER_SET. JMS_IBM_CHARACTER_SET, podczas wysyłania komunikatu musi być to liczbowy identyfikator kodowanego zestawu znaków[4].

Fragmenty kodu w programie “Wysyłanie i odbieranie JMSTextmessage” na stronie 178 wysyłają dwa komunikaty. Jeden z nich 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 korzystają ze strony kodowej zdefiniowanej we właściwości komunikatu JMS_IBM_CHARACTER_SET. Strona kodowa jest odwzorowywana z programu MQRFH2.CodedCharacterSetId , chyba że komunikat jest komunikatem typu MQi nie zawiera MQRFH2. Jeśli komunikat jest komunikatem typu MQ, bez elementu MQRFH2, strona kodowa jest odwzorowywana z produktu MQMD.CodedCharacterSetId.

Fragment kodu w programie Rysunek 29 na stronie 178 odbiera komunikat, który został wysłany do miejsca docelowego. Tekst w komunikacie jest przekształcany ze strony kodowej IBM037 z powrotem na kod Unicode.

Uwaga: Prostym sposobem sprawdzenia, czy tekst jest przekształcany na kodowany zestaw znaków 37, jest użycie programu IBM MQ Explorer. Należy przejrzeć kolejkę i wyświetlić właściwości komunikatu przed jego pobraniem.

[4] Podczas odbierania komunikatu JMS_IBM_CHARACTER_SET jest nazwą strony kodowej Java Charset .

Przeciwstawny fragment kodu w programie Rysunek 28 na stronie 178 z niepoprawnym fragmentem kodu w produkcie Rysunek 25 na stronie 176. W niepoprawnym fragmencie kodu łańcuch tekstowy jest przekształcany dwa razy, raz przez aplikację, a następnie ponownie przez program 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` przekształca tekst z kodu Unicode na wartość 1208, UTF-8. Łańcuch tekstowy jest poprzedzony o długości 2 bajtów. Maksymalna długość łańcucha tekstowego to 65534 bajtów. Metoda `readUTF` odczytuje element w komunikacie zapisanej w metodzie `writeUTF` . Odczytuje dokładnie liczbę bajtów zapisanych w metodzie `writeUTF` .

Liczbowy

Domyślnym kodowaniem liczbowym dla miejsca docelowego jest `Native`. Stała kodowana `Native` dla Java ma wartość 273, `x'00000111'`, która jest taka sama dla wszystkich platform. Po odebraniu numery w komunikacie są poprawnie przekształcane w liczbowe podstawowe elementy Java . Transformacja korzysta z kodowania zdefiniowanego w komunikacie i typu zwróconego przez metodę odczytu.

Metoda wysyłania przekształca liczby dodawane do komunikatu przez `set` i `write` do kodowania liczbowego zdefiniowanego dla miejsca docelowego. Kodowanie docelowe może zostać przesłonięte dla komunikatu przez aplikację ustawiające właściwość komunikatu `JMS_IBM_ENCODING` . na przykład:

```
message.setIntProperty(WMQConstants.JMS_IBM_ENCODING,
WMQConstants.WMQ_ENCODING_INTEGER_REVERSED);
```

Metody numeryczne `get` i `read` przekształcają liczby w komunikacie z kodowania liczbowego zdefiniowanego w komunikacie. Te liczby są przekształcane do typu określonego za pomocą metody `read` lub `get` . Patrz sekcja Właściwość ENCODING. Metody używają kodowania zdefiniowanego w produkcie `JMS_IBM_ENCODING`. Kodowanie jest odwzorowywane z programu `MQRFH2.Encoding` , o ile komunikat nie jest komunikatem typu MQi nie zawiera `MQRFH2`. Jeśli komunikat jest komunikatem typu MQ, w którym nie ma `MQRFH2`, metody używają kodowania zdefiniowanego w produkcie `MQMD.Encoding`.

W przykładzie w produkcie Rysunek 30 na stronie 178 jest wyświetlana aplikacja koduje liczbę w formacie docelowym i wysyła ją w pliku `JMSStreamMessage`. Należy porównać przykład z Rysunek 30 na stronie 178 z przykładem w Rysunek 31 na stronie 179. Różnica polega na tym, że wartość `JMS_IBM_ENCODING` musi być ustawiona w `JMSBytesMessage`.

Uwaga: Prostym sposobem sprawdzenia, czy numer jest zakodowany poprawnie, jest użycie programu IBM MQ Explorer. Należy przejrzeć kolejkę i wyświetlić właściwości komunikatu przed jego konsumowaniu.

Inne

The `boolean` methods encode `true` and `false` as `x'01'` and `x'00'` in a `JMSByteMessage`, `JMSStreamMessage`, and `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 obiektu zawijają podstawowe typy jako obiekty. Typy liczbowe i tekstowe są kodowane lub przekształcane tak, jakby typy podstawowe zostały odczytane lub zapisane przy użyciu metod numerycznych i tekstowych.

Brak

Metody `readByte`, `readBytes`, `readUnsignedByte`, `writeByte`i `writeBytes` otrzymują lub umieszczają pojedyncze bajty lub tablice bajtów, między aplikacją a komunikatem bez konwersji. Metody `readChar` i `writeChar` dostają i umieszczają 2 bajtowe znaki Unicode między aplikacją a komunikatem bez konwersji.

Za pomocą metod `readBytes` i `writeBytes` aplikacja może przeprowadzić konwersję własnego punktu kodowego, tak jak w przypadku produktu “Wysyłanie i odbieranie tekstu w JMSBytesMessage” na stronie 179.

Produkt IBM MQ nie wykonuje żadnej konwersji strony kodowej w kliencie, ponieważ jest to komunikat `JMSBytesMessage`, a także z uwagi na to, że używane są metody `readBytes` i `writeBytes` . Jeśli jednak bajty reprezentują tekst, należy się upewnić, że strona kodowa używana przez aplikację jest zgodna z kodowanym zestawem znaków miejsca docelowego. Komunikat może zostać ponownie przekształcony przez wyjście konwersji menedżera kolejek. Inną możliwością jest to, że program kliencki odbierający produkt JMS może być zgodny z konwencją przekształcania tablic bajtowych reprezentujących tekst w komunikacie na łańcuchy lub znaki za pomocą właściwości `JMS_IBM_CHARACTER_SET` w komunikacie.

W tym przykładzie klient używa docelowego zestawu znaków określonego dla jego konwersji:

```
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 w właściwości `JMS_IBM_CHARACTER_SET` komunikatu. IBM MQ classes for Java użyj `JMS_IBM_CHARACTER_SET` , aby ustawić pole `CodedCharacterSetId` we właściwościach JMS w `MQRFH2`lub w deskryptorze komunikatu, `MQMD`:

```
String codePage = CCSID.getCodepage(37);
message.setIntProperty(WMQConstants.JMS_IBM_CHARACTER_SET, codePage);
5
```

Jeśli tablica bajtów jest zapisywana w `JMSStringMessage` lub `JMSMapMessage`, program IBM MQ classes for JMS nie wykonuje 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, które punkty kodowe mają zostać odczytane i zapisane w komunikacie. Kod w programie Rysunek 26 na stronie 177 jest zgodny z konwencją używania docelowego zestawu znaków kodowanych. Jeśli łańcuch zostanie utworzony przy użyciu domyślnego zestawu znaków dla maszyny JVM, zawartość bajtów będzie zależeć od platformy. A JVM on Windows typically has a default `Charset` of `windows-1252`, and UNIX, `UTF-8`. Wymiana między Windows a UNIX wymaga wybrania jawnej strony kodowej w celu wymiany tekstu w postaci 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

[5] `SetStringProperty(WMQConstants.JMS_IBM_CHARACTER_SET, codePage)` currently accepts only numeric character set identifiers.

Wysyłanie i odbieranie JMSTextmessage

Komunikat tekstowy nie może zawierać tekstu w różnych zestawach znaków. W przykładzie przedstawiono tekst w różnych zestawach znaków, które są wysyłane 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 wiadomość tekstową 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

```
TextMessage tmi = (TextMessage)consumer.receive();
System.out.println(tmi.getText());
...
Sent in the character set defined for the destination
```

Rysunek 29. Odbierz wiadomość tekstową

Przykłady kodowania

Przykłady pokazujące liczbę wysyłanej w kodowaniu definicji dla miejsca docelowego. Należy zauważyć, że należy ustawić właściwość JMS_IBM_ENCODING dla JMSBytesMessage na 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 numeru przy użyciu kodowania miejsca docelowego w produkcie 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 numeru przy użyciu kodowania miejsca docelowego w produkcie JMSBytesMessage

Wysyłanie i odbieranie tekstu w JMSBytesMessage

Kod w pliku Rysunek 32 na stronie 179 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 179. 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 przy użyciu aplikacji innej niżJMS
Wykonaj kroki proponowane w tym zadaniu, aby zaprojektować i zbudować wyjście konwersji danych, a także aplikację kliencką JMS , która może wymieniać komunikaty z aplikacją inną niżJMS przy użyciu

produktu JMSBytesMessage. Wymiana sformatowanego komunikatu z aplikacją inną niżJMS może odbywać się bez wywoływania wyjścia konwersji danych lub bez niej.

Odsyłacze pokrewne

Typy komunikatów i konwersje produktu JMS
Wybór typu komunikatu wpływa na podejście do konwersji komunikatów. Interakcja konwersji komunikatów i typu komunikatu jest opisana dla typów komunikatów JMS , JMSObjectMessage, JMSTextMessage, JMSMapMessage, JMSStreamMessagei 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, Encodingi 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_CONVERSIONna wartość WMQ_RECEIVE_CONVERSION_QMGRlub 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 181,

```
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
Na liście znajdują się metody używane do konwersji i kodowania komunikatów klienta JMS , z przykładami kodu każdego typu konwersji.

“Wywoływanie wyjścia konwersji danych” na stronie 1083
Wyjście konwersji danych to wyjście napisane przez użytkownika, które odbiera sterowanie podczas przetwarzania wywołania MQGET.

Zadania pokrewne

Wymiana sformatowanego rekordu przy użyciu aplikacji innej niżJMS
Wykonaj kroki proponowane w tym zadaniu, aby zaprojektować i zbudować wyjście konwersji danych, a także aplikację kliencką JMS , która może wymieniać komunikaty z aplikacją inną niżJMS przy użyciu produktu JMSBytesMessage. Wymiana sformatowanego komunikatu z aplikacją inną niżJMS może odbywać się bez wywoływania wyjścia konwersji danych lub bez niej.

Odsyłacze pokrewne

Typy komunikatów i konwersje produktu JMS

Wybór typu komunikatu wpływa na podejście do konwersji komunikatów. Interakcja konwersji komunikatów i typu komunikatu jest opisana dla typów komunikatów JMS , `JMSObjectMessage`, `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessage`i `JMSBytesMessage`.

Wymiana sformatowanego rekordu przy użyciu aplikacji innej niżJMS
Wykonaj kroki proponowane w tym zadaniu, aby zaprojektować i zbudować wyjście konwersji danych, a także aplikację kliencką JMS , która może wymieniać komunikaty z aplikacją inną niżJMS przy użyciu produktu `JMSBytesMessage`. Wymiana sformatowanego komunikatu z aplikacją inną niżJMS może odbywać się bez wywoływania wyjścia konwersji danych lub bez niej.

Zanim rozpoczniesz

Użytkownik może zaprojektować prostsze rozwiązanie do wymiany komunikatów za pomocą aplikacji innej niżJMS przy użyciu `JMSTextMessage`. Wyeliminuj tę możliwość przed krokami kroków w tym zadaniu.

O tym zadaniu

Klient JMS jest łatwiejszy do pisania, jeśli nie jest zaangażowany w szczegóły dotyczące formatowania komunikatów produktu JMS wymienianych z innymi klientami JMS . Jeśli typem komunikatu jest `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessage`lub `JMSObjectMessage`, program IBM MQ będzie wyglądał po szczegółach formatowania komunikatu. Program IBM MQ zajmuje się różnicami w stronach kodowych i kodowaniem liczbowym na różnych platformach.

Tych typów komunikatów można używać do wymiany komunikatów z aplikacjami innymi niżJMS . Aby to zrobić, należy zrozumieć, w jaki sposób te komunikaty są tworzone przez produkt IBM MQ classes for JMS. Użytkownik może mieć możliwość zmodyfikowania aplikacji innej niżJMS w celu zinterpretowania komunikatów; patrz “Odwzorowywanie komunikatów produktu JMS na komunikaty produktu IBM MQ” na stronie 142.

Zaletą korzystania z jednego z tych typów komunikatów jest programowanie klienta JMS nie zależy od typu aplikacji, z którą są wymieniane komunikaty. 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 mieszaninę niesformatowanych danych, tekstu i liczb. Wykonaj kroki opisane w tej czynności, a następnie klienta JMS w produkcie “Zapisywanie klas w celu hermetyzowania układu rekordów w JMSBytesMessage” na stronie 185, jako punkt początkowy budowania klienta produktu JMS , który może wymieniać sformatowane rekordy z aplikacjami innymi niżJMS .

Procedura

1. Zdefiniuj układ rekordu lub użyj jednej z predefiniowanych klas nagłówków produktu IBM MQ .

   Informacje na temat obsługi predefiniowanych nagłówków produktu IBM MQ można znaleźć w sekcji Obsługa nagłówków komunikatów produktu IBM MQ.

   Rysunek 37 na stronie 183 jest przykładem zdefiniowanego przez użytkownika układu rekordu 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 wyjście konwersji danych.

   Aby wypróbować przykład w programie “Zapisywanie klas w celu hermetyzowania układu rekordów w JMSBytesMessage” na stronie 185, należy podać nazwę wyjścia konwersji danych `MYRECORD`.
3. Klasy Java można zapisać w celu hermetyzowania układu rekordu oraz wysyłania i odbierania akt. Dostępne są dwa sposoby podejścia:

   - Napisz klasę do tego odczytu i zapisze `JMSBytesMessage` , który zawiera rekord; patrz “Zapisywanie klas w celu hermetyzowania układu rekordów w JMSBytesMessage” na stronie 185.

- Napisz klasę rozszerzająca `com.ibm.mq.header.Header` , aby zdefiniować strukturę danych dla rekordu. Patrz sekcja Tworzenie klas dla nowych typów nagłówków.

4. Zdecyduj, jaki kodowany zestaw znaków ma być używany do wymiany komunikatów.

   Więcej informacji na ten temat zawiera sekcja Wybieranie podejścia do konwersji komunikatów: odbiorca sprawia, że jest on dobry.

5. Skonfiguruj miejsce docelowe w celu wymiany komunikatów typu MQ, bez nagłówka JMS `MQRFH2` .

   Zarówno miejsce docelowe wysyłania, jak i odbierania musi być skonfigurowane do wymiany komunikatów typu MQ. Tego samego miejsca docelowego można użyć zarówno do wysyłania, jak i do odbierania.

   Aplikacja może przesłonić właściwość treści komunikatu docelowego:

   ```
   ((MQDestination)destination).setMessageBodyStyle(WMQConstants.WMQ_MESSAGE_BODY_MQ);
   ```

   Przykład w produkcie "Zapisywanie klas w celu hermetyzowania układu rekordów w JMSBytesMessage" na stronie 185 przesłania właściwość treści komunikatu docelowego, upewniając się, że wysyłany jest komunikat w stylu MQ.

6. Testowanie rozwiązania za pomocą aplikacji JMS i innych 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.h`) w produkcie Rysunek 38 na stronie 184. W przypadku modyfikacji program `amqsgetc0.c` otrzymuje komunikat wysłany przez przykładowego klienta JMS , `TryMyRecord.java` ; patrz "Zapisywanie klas w celu hermetyzowania układu rekordów w JMSBytesMessage" na stronie 185.
   - Przykładowy program do przeglądania IBM MQ , `amqsbcg0.c`, jest przydatny do sprawdzenia zawartości nagłówka komunikatu, nagłówka JMS , `MQRFH2`oraz treści komunikatu.
   - Program **`rfhutil`** , który jest wcześniej dostępny w pliku SupportPac IH03, umożliwia przechwytywanie komunikatów testowych i przechowywanie ich w plikach, a następnie ich użycie w celu kierowania przepływów komunikatów. Komunikaty wyjściowe mogą być również odczytywane i wyświetlane w różnych formatach. Formaty te obejmują dwa typy kodu XML, a także dopasowanie do struktury copybook języka COBOL. Dane mogą być w formacie EBCDIC lub ASCII. Nagłówek RFH2 może zostać dodany do komunikatu przed wysłaniem komunikatu.

   W przypadku próby odebrania komunikatów za pomocą zmodyfikowanego przykładowego programu `amqsgetc0.c` i uzyskania błędu o kodzie przyczyny `2080`należy sprawdzić, czy komunikat ma `MQRFH2`. W przypadku modyfikacji 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 produktu 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 tak, aby korzystało z 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 print,

  1. Od:

```
buffer[messlen] = '\0';            /* add terminator          */
printf("message <%s>\n", buffer);
```

  2. to:

```
/* 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

Na liście znajdują się metody używane do konwersji i kodowania komunikatów klienta JMS , z przykładami kodu 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.

Odsyłacze pokrewne

Typy komunikatów i konwersje produktu JMS
Wybór typu komunikatu wpływa na podejście do konwersji komunikatów. Interakcja konwersji komunikatów i typu komunikatu jest opisana dla typów komunikatów JMS , `JMSObjectMessage`, `JMSTextMessage`, `JMSMapMessage`, `JMSStreamMessage`i `JMSBytesMessage`.

Program narzędziowy do tworzenia kodu wyjścia konwersji

Zapisywanie klas w celu hermetyzowania układu rekordów w JMSBytesMessage
Celem tego zadania jest eksplorowanie, na przykład, sposobu łączenia konwersji danych i stałego układu rekordów w `JMSBytesMessage`. W ramach zadania tworzone są niektóre klasy produktu Java w celu wymiany przykładowej struktury rekordów w `JMSBytesMessage`. Istnieje możliwość zmodyfikowania przykładu w celu zapisania klas w celu wymiany innych struktur akt.

`JMSBytesMessage` jest najlepszym wyborem typu komunikatu JMS w celu wymiany rekordów mieszanych typów danych z programami innymi niżJMS . Nie ma żadnych dodatkowych danych wstawianych do treści komunikatu przez dostawcę JMS . Dlatego najlepszym wyborem typu komunikatu jest użycie, jeśli program kliencki JMS współdziała z istniejącym programem IBM MQ . Główne wyzwanie związane z używaniem `JMSBytesMessage` jest zgodne z kodowaniem i zestawem znaków oczekiwanym przez inny program. Rozwiązaniem jest utworzenie klasy, która hermetyzuje rekord. Klasa, która hermetyzuje odczytywanie i zapisywanie `JMSBytesMessage`dla konkretnego typu akt, ułatwia wysyłanie i odbieranie rekordów w formacie ustalonym w programie JMS . Przechwytywanie ogólnych aspektów interfejsu w klasie abstrakcyjnej, wiele z rozwiązań może być ponownie wykorzystane dla różnych formatów rekordu. Różne formaty rekordów mogą być implementowane w klasach, które rozszerzają abstrakcyjną klasę rodzajową.

Alternatywnym podejściem jest rozszerzenie klasy `com.ibm.mq.headers.Header` . Klasa `Header` ma metody, takie jak `addMQLONG`, w celu budowania formatu rekordu w bardziej deklaratywny sposób. Wadą korzystania z klasy `Header` jest pobieranie i ustawianie atrybutów przy użyciu bardziej skomplikowanego interfejsu interpretacyjnego. Oba podejścia powodują w dużej ilości tyle samo kodu aplikacji.

Program `JMSBytesMessage` może hermetyzować tylko jeden format, oprócz `MQRFH2`, w jednym komunikacie, o ile każdy rekord nie używa tego samego formatu, kodowanego zestawu znaków i kodowania. Format, kodowanie i zestaw znaków `JMSBytesMessage` to właściwości wszystkich komunikatów następujących po `MQRFH2`. Przykład ten jest zapisany przy założeniu, że `JMSBytesMessage` zawiera tylko jeden rekord użytkownika.

Zanim rozpoczniesz

1. Poziom umiejętności: użytkownik musi być zaznajomiony z programowaniem Java i programem JMS. Nie są dostępne żadne instrukcje dotyczące konfigurowania środowiska programistycznego produktu Java . Zaleca się napisanie programu do wymiany serwerów `JMSTextMessage`, `JMSStreamMessage`lub `JMSMapMessage`. Użytkownik może następnie zapoznać się z różnicami w wymianie komunikatu przy użyciu `JMSBytesMessage`.
2. W tym przykładzie wymagane jest IBM WebSphere MQ 7.0.
3. Przykład został utworzony przy użyciu perspektywy Java środowiska roboczego Eclipse . Wymaga ona środowiska JRE 6.0 lub nowszego. Aby utworzyć i uruchomić klasy produktu Java , można użyć perspektywy Java w programie IBM MQ Explorer. Alternatywnie można użyć własnego środowiska programistycznego produktu Java .
4. Korzystanie z programu IBM MQ Explorer powoduje skonfigurowanie środowiska testowego oraz debugowanie, prostsze niż korzystanie z programów narzędziowych wiersza komend.

O tym zadaniu

Użytkownik jest prowadzony przez utworzenie dwóch klas: RECORD i MyRecord. Razem te dwie klasy hermetyzują rekord o stałym formacie. Mają one metody pobierania i ustawiania atrybutów. Metoda get odczytuje rekord z JMSBytesMessage , a metoda put zapisuje rekord w JMSBytesMessage.

Zadaniem tego zadania nie jest utworzenie klasy jakości produkcyjnej, którą można ponownie wykorzystać. Można użyć przykładów w zadaniu, aby rozpocząć działanie na własnych zajęciach. Celem tego zadania jest udostępnienie uwag dotyczących wskazówek, przede wszystkim dotyczących używania zestawów znaków, formatów i kodowania, w przypadku korzystania z produktu JMSBytesMessage. Każdy krok tworzenia klas jest wyjaśniony, a aspekty korzystania z produktu JMSBytesMessage, które czasami są pomijane, są opisane.

Klasa RECORD jest abstrakcyjna i definiuje niektóre wspólne pola dla rekordu użytkownika. Wspólne pola są modelowane na standardowym układzie nagłówka IBM MQ , w którym znajduje się program catcher, wersja i pole długości. Pola kodowania, zestawu znaków i formatu, które znajdują 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 dosłownie rozszerzanie rekordu z dodatkowymi polami użytkownika. Program JMSBytesMessage, utworzony przez klasy, może być przetwarzany przez wyjście konwersji danych menedżera kolejek.

Produkt “Klasy używane do uruchamiania przykładu” na stronie 192 zawiera pełną listę produktów RECORD i MyRecord. Zawiera również listy dodatkowych klas "rusztowań" w celu przetestowania produktów 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 jednej klasie. Jego interfejs spełnia tylko potrzeby testowania klas RECORD i MyRecord . Nie jest to ustalony wzorzec projektowania dla tworzenia aplikacji produktu JMS .

Uwaga: Klasa Endpoint zawiera ten 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 wersji V7.0, konwersja do V7.0.1.4 menedżera kolejek jest domyślnie włączona, a ten wiersz kodu powoduje wystąpienie błędu.

MyProducer i MyConsumer
Klasy, które rozszerzają EndPointi tworzą MessageConsumer i MessageProducer, połączone i gotowe do akceptowania żądań.

Razem wszystkie klasy składają się na kompletną aplikację, z którą można zbudować i eksperymentować, aby zrozumieć, jak używać konwersji danych w JMSBytesMessage.

Procedura

1. Utwórz klasę abstrakcyjną, aby hermetyzować standardowe pola w nagłówku IBM MQ , przy użyciu konstruktora domyślnego. Później rozszerz klasę, aby dostosować nagłówek do 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 `nextFormats`ą wymienione w kolejności, w jakiej są one określone w standardowym nagłówku komunikatu produktu IBM MQ .

b. Atrybuty, `format`, `messageEncoding`i `messageCharset`, opisują sam nagłówek i nie są częścią nagłówka.

c. Należy zdecydować, czy ma być przechowywany identyfikator kodowanego zestawu znaków, czy zestaw znaków rekordu. Produkt Java używa zestawów znaków, a komunikaty produktu IBM MQ używają identyfikatorów kodowanego zestawu znaków. Przykładowy kod używa zestawów znaków.

d. `int` jest serializowany do `MQLONG` przez 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 produktu RECORD na podstawie `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 zastępują one wartości ustawione dla miejsca docelowego. Produkt `format` nie został

zaktualizowany. W przykładzie nie jest sprawdzane żadne błędy. Jeśli wywołano konstruktor `Record(BytesMessage)` , przyjmuje się, że `JMSBytesMessage` jest komunikatem typu `RECORD` . Linia "`setStructID(new String(structID, getMessageCharset()))`" ustawia łapacz oczu.

b. Wiersze kodu, które wypełniają pola z deserializacją metody w komunikacie, w kolejności aktualizowania wartości domyślnych ustawionych 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. Program `MyProducer` hermetykuje JMS `Connection`, `Destination`, `Session`i `MessageProducer` w jednej klasie. MyConsumer, używany później, hermetykuje JMS `Connection`, `Destination`, `Session`i `MessageConsumer` w jednej klasie.

b. W przypadku partycji `JMSBytesMessage`, jeśli kodowanie jest inne niż `Native`, kodowanie musi być ustawione w komunikacie. Kodowanie docelowe jest kopiowane do atrybutu kodowania komunikatów `JMS_IBM_CHARACTER_SET`i zapisywane jako atrybut klasy `RECORD` .

   i) Program "`setMessageEncoding(myProducer.getEncoding());`" wywołuje "`(((MQDestination) destination).getIntProperty(WMQConstants.WMQ_ENCODING));`" , aby uzyskać kodowanie miejsca docelowego.

   ii) "`Bytes.setIntProperty(WMQConstants.JMS_IBM_ENCODING, getMessageEncoding());`" ustawia kodowanie komunikatów.

c. Zestaw znaków używany do transformowania tekstu w bajty jest uzyskiwany z miejsca docelowego i zapisywany jako atrybut klasy `RECORD` . Nie jest on ustawiony w komunikacie, ponieważ nie jest używany przez produkt IBM MQ classes for JMS podczas zapisywania `JMSBytesMessage`.

Wywołania "`messageCharset = myProducer.getCharset();`"

```
public String getCharset() throws UnsupportedEncodingException,
        JMSException {
    return CCSID.getCodepage(getCCSID());
}
```

Pobiera on zestaw znaków Java z identyfikatora kodowanego zestawu znaków.

" `CCSID.getCodepage(ccsid)` " znajduje się w pakiecie `com.ibm.mq.headers`. `ccsid` jest uzyskiwane z innej metody w produkcie `MyProducer`, która wysyła zapytania 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 go na IBM MQ MQI client. Może być konieczne pominięcie tej linii kodu, ponieważ powoduje to wystąpienie błędu konfiguracji administracyjnej.

"`myProducer.setMQClient(true);`" wywołuje:

```
((MQDestination) destination).setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ); }
if (!getMQDest()) setMQBody();
```

Kod ma skutek uboczny ustawienia stylu treści IBM MQ na nieokreślony, jeśli musi 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 ustawiać pól `MQMD` ręcznie.

Istnieje metoda ręcznego ustawiania właściwości deskryptora komunikatu. Używa on właściwości produktu `JMS_IBM_MQMD_*` . Aby ustawić właściwości produktu `JMS_IBM_MQMD_*` , należy ustawić właściwość docelową `WMQ_MQMD_WRITE_ENABLED` :

```
((MQDestination)destination).setMQMDWriteEnabled(true);
```

Aby odczytać właściwości, należy ustawić właściwość miejsca docelowego `WMQ_MQMD_READ_ENABLED`.

`JMS_IBM_MQMD_*` należy używać tylko wtedy, gdy użytkownik zapełni kontrolę nad całym ładunkiem komunikatu. W przeciwieństwie do właściwości `JMS_IBM_*` , właściwości `JMS_IBM_MQMD_*` nie sterują sposobem konstruowania komunikatu JMS przez program IBM MQ classes for JMS . Możliwe jest utworzenie właściwości deskryptora komunikatu, które powodują konflikt z właściwościami komunikatu produktu JMS .

e. Wiersze kodu, które ukończy metodę, serializują atrybuty w klasie jako pola w komunikacie.

Atrybuty łańcuchowe są dopełniane spacjami. Łańcuchy są przekształcane w bajty przy użyciu zestawu znaków zdefiniowanego dla rekordu, a następnie 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 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;
```

6. Utwórz klasę, aby rozszerzyć klasę RECORD w celu uwzględnienia dodatkowych pól. Uwzględnij 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 chwytnik do oczu, format i długość nagłówka.

7. Utwórz lub wygeneruj procedury pobierające i ustawiające.

   a) Utwórz metody 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 produktu MyRecord na podstawie 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 składają się na standardowy szablon komunikatu, są odczytyane najpierw przez klasę RECORD .

b. Tekst recordData jest przekształcany w produkt String przy użyciu właściwości zestawu znaków komunikatu.

9. Utwórz metodę statyczną, aby pobrać komunikat od konsumenta i utworzyć nową instancję produktu 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 przypadku zapierania, konstruktor MyRecord(BytesMessage) jest wywoływany z metody get static. Zwykle użytkownik może oddzielić odbieranie komunikatu od utworzenia nowej instancji produktu MyRecord .

10. Utwórz metodę put, aby dodać pola klienta do 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. Metoda wywołuje w kodzie serializację atrybutów w klasie MyRecord jako pola w komunikacie.

- Atrybut `recordData` `String` jest dopełniany spacjami, przekształcony w bajty przy użyciu zestawu znaków zdefiniowanego dla rekordu i obcięty do długości pól `RecordData` .

11. Zakończ klasę, dodając instrukcje include.

```
 package com.ibm.mq.id;
import java.io.IOException;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import com.ibm.mq.headers.MQDataException;
```

Wyniki

Wyniki:

- Wyniki działania klasy `TryMyRecord` :
  - Wysyłanie komunikatu z kodowanego zestawu znaków 37 i użycie 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, a nie 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 546 CCSID IBM037
```

- Wynikiem modyfikacji klasy `TryMyRecord` nie jest odebranie komunikatu, a następnie odbieranie go za pomocą zmodyfikowanej próbki produktu `amqsget0.c` . Zmodyfikowana próbka akceptuje sformatowany rekord; patrz Rysunek 38 na stronie 184 w “Wymiana sformatowanego rekordu przy użyciu aplikacji innej niżJMS” na stronie 182.
  - Wysyłanie komunikatu z kodowanego zestawu znaków 37 i użycie 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, a nie przy użyciu 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
```

W celu wypróbowania przykładu i eksperymentu z różnymi stronami kodowymi i wyjściem konwersji danych. Utwórz klasy produktu Java , skonfiguruj produkt IBM MQi uruchom program główny `TryMyRecord` . patrz Rysunek 39 na stronie 193.

1. Skonfiguruj produkt IBM MQ i JMS , aby uruchomić przykład. Instrukcje są przeznaczone do 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 bin produktu JMS .

   Program administracyjny JMS musi być uruchomiony z tego miejsca. 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 produktu JMS .

   ```
   JMSAdmin < JMSQM1Q1.txt
   ```

2. Użytkownik może tworzyć, zmieniać i przeglądać definicje utworzone za pomocą programu IBM MQ Explorer.
3. Uruchom program `TryMyRecord`.

Klasy używane do uruchamiania przykładu

Klasy wymienione w rysunkach Rysunek 39 na stronie 193 do Rysunek 44 na stronie 197 są również dostępne w pliku skompresowanym; należy pobrać plik jm25529_.zip lub jm25529_.tar.gz.

```
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());
    }
}
```

Rysunek 39. TryMyRecord

```
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; }
}
```

Rysunek 40. RECORD

```
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; }
}
```

Rysunek 41. MyRecord

```
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); }
    }
```

Rysunek 42. EndPoint

```
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); }
}
```

Rysunek 43. MyProducer

```
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();  }
}
```

Rysunek 44. MyConsumer

Tworzenie i konfigurowanie fabryk połączeń i miejsc docelowych w aplikacji IBM MQ classes for JMS

Aplikacja IBM MQ classes for JMS może tworzyć fabryki połączeń i miejsca docelowe, pobierając je jako administrowane obiekty z przestrzeni nazw Java Naming and Directory Interface (JNDI), korzystając z rozszerzeń IBM JMS lub za pomocą rozszerzeń produktu 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 są punktami startów w przepływie logiki aplikacji JMS . Aplikacja korzysta z obiektu ConnectionFactory w celu utworzenia połączenia z serwerem przesyłania komunikatów i używa obiektu Queue lub Topic jako elementu docelowego do wysyłania komunikatów do lub do źródła, z którego mają być odbierane komunikaty. W związku z tym 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 poprzez ustawienie jednej lub większej liczby jej właściwości.

Podsumowując, aplikacja może tworzyć i konfigurować fabryki połączeń i miejsca docelowe w jeden z następujących sposobów:

Używanie interfejsu JNDI do pobierania administrowanych obiektów

Administrator może użyć narzędzia administracyjnego IBM MQ JMS w sposób opisany w sekcji Konfigurowanie obiektów przy użyciu narzędzia administracyjnego JMSlub IBM MQ Explorer w sposób opisany w sekcji Konfigurowanie obiektów produktu JMS przy użyciu produktu IBM MQ Explorerw celu utworzenia i skonfigurowania fabryk połączeń i miejsc docelowych jako obiektów administrowanych w przestrzeni nazw JNDI. Aplikacja może następnie pobrać administrowane obiekty z przestrzeni nazw JNDI. Po pobraniu administrowanego obiektu aplikacja może, jeśli jest to wymagane, ustawić

lub zmienić jedną lub więcej jej właściwości, korzystając z rozszerzeń IBM JMS lub rozszerzeń IBM MQ JMS .

Korzystanie z rozszerzeń IBM JMS

Aplikacja może używać rozszerzeń 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 jego właściwości. Alternatywnie aplikacja może użyć identyfikatora URI (Uniform Resource Identifier) w celu określenia jednej lub większej liczby właściwości miejsca docelowego podczas tworzenia miejsca docelowego.

Korzystanie z rozszerzeń IBM MQ JMS

Aplikacja może również używać rozszerzeń IBM MQ JMS do dynamicznego tworzenia fabryk połączeń i miejsc docelowych w czasie wykonywania. Aplikacja korzysta z dostarczonych konstruktorów w celu utworzenia fabryk połączeń i miejsc docelowych. Po utworzeniu fabryki połączeń lub miejsca docelowego aplikacja może użyć metod obiektu w celu ustawienia jego właściwości. Alternatywnie aplikacja może użyć identyfikatora URI, aby określić jedną lub więcej właściwości miejsca docelowego podczas tworzenia miejsca docelowego.

Zadania pokrewne

Konfigurowanie zasobów produktu JMS

Korzystanie z interfejsu JNDI do pobierania obiektów administrowanych w aplikacji JMS

Aby pobrać obiekty administrowane z przestrzeni nazw interfejsu Java Naming and Directory Interface (JNDI), aplikacja JMS musi utworzyć kontekst początkowy, a następnie użyć metody lookup () do pobrania obiektów.

Zanim aplikacja będzie mogła pobierać administrowane obiekty z przestrzeni nazw JNDI, administrator musi najpierw utworzyć administrowane obiekty. Administrator może użyć narzędzia administracyjnego IBM MQ JMS lub IBM MQ Explorer w celu utworzenia i obsługi obiektów administrowanych w przestrzeni nazw JNDI. Więcej informacji na ten temat zawiera sekcja Konfigurowanie fabryk połączeń i miejsc docelowych w przestrzeni nazw JNDI.

Serwer aplikacji, zwykle udostępnia własne repozytorium dla administrowanych obiektów oraz własne narzędzia do tworzenia i obsługi obiektów.

Aby pobrać obiekty administrowane z przestrzeni nazw JNDI, aplikacja musi najpierw utworzyć kontekst początkowy, tak jak pokazano to w poniższym przykładzie:

```
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 (Uniform Resource Locator) usługi katalogowej. Adres 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 powodować wystąpienie błędu LDAP 84. Aby rozwiązać ten problem, przed wywołaniem funkcji InitialDirContext () należy wstawić następujący wiersz kodu:

```
environment.put(Context.REFERRAL, "throw");
```

Po uzyskaniu kontekstu początkowego aplikacja może pobierać administrowane obiekty 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 udostępniana przez firmę Oracle Corporation.

Zadania pokrewne

Konfigurowanie obiektów produktu JMS przy użyciu produktu IBM MQ Explorer

Konfigurowanie obiektów produktu JMS przy użyciu narzędzia administracyjnego

Konfigurowanie zasobów produktu JMS w produkcie WebSphere Application Server

Korzystanie z rozszerzeń IBM JMS

Produkt IBM MQ classes for JMS zawiera zestaw rozszerzeń do interfejsu API produktu JMS o nazwie rozszerzenia IBM JMS . Aplikacja może używać tych rozszerzeń do dynamicznego tworzenia fabryk połączeń i miejsc docelowych w czasie wykonywania, a także do ustawiania właściwości obiektów produktu IBM MQ classes for JMS . Rozszerzenia mogą być używane z dowolnym dostawcą przesyłania komunikatów.

Rozszerzenia IBM JMS są zestawem interfejsów i klas w następujących pakietach:

- com.ibm.msg.client.jms
- com.ibm.msg.client.services

Pakiety te można znaleźć w programie `com.ibm.mqjms.jar` , który znajduje się w *MQ_INSTALLATION_PATH*`/java/lib`.

Rozszerzenia te udostępniają następujące funkcje:

- Mechanizm fabryczny do dynamicznego tworzenia fabryk połączeń i miejsc docelowych w czasie wykonywania, zamiast pobierania ich jako administrowanych obiektów z przestrzeni nazw interfejsu Java Naming and Directory Interface (JNDI).
- Zestaw metod ustawiania właściwości obiektów produktu IBM MQ classes for JMS .
- Zestaw klas wyjątków z metodami uzyskiwania szczegółowych informacji na temat problemu
- Zestaw metod sterowania śledzeniem
- Zestaw metod uzyskiwania informacji o wersji na temat produktu IBM MQ classes for JMS

Jeśli chodzi o dynamiczne tworzenie fabryk połączeń i miejsc docelowych w czasie wykonywania, a także ustawianie i pobieranie ich właściwości, rozszerzenia IBM JMS udostępniają alternatywny zestaw interfejsów do rozszerzeń IBM MQ JMS . Jednak rozszerzenia IBM MQ JMS są specyficzne dla dostawcy przesyłania komunikatów produktu IBM MQ , ale rozszerzenia IBM JMS nie są specyficzne dla produktu IBM MQ i mogą być używane z dowolnym dostawcą przesyłania komunikatów w architekturze warstwowej opisanej w sekcji Klasy IBM MQ dla architektury JMS.

Interfejs com.ibm.msg.client.wmq.WMQConstants zawiera definicje stałych, które mogą być używane przez aplikację podczas ustawiania właściwości obiektów produktu IBM MQ classes for JMS przy użyciu rozszerzeń IBM JMS . Interfejs zawiera stałe dla stałych dostawców przesyłania komunikatów produktu IBM MQ i JMS , które są niezależne od dowolnego dostawcy przesyłania komunikatów.

W poniższych przykładach kodu przyjęto założenie, że zostały uruchomione następujące instrukcje importu:

```
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 utworzyć fabryki połączeń i miejsca docelowe przy użyciu rozszerzeń IBM JMS , musi najpierw utworzyć obiekt fabryki JmsFactory. Aby utworzyć obiekt fabryki JmsFactory, aplikacja wywołuje metodę getInstance() klasy fabryki JmsFactory, tak jak przedstawiono to w poniższym przykładzie:

```
JmsFactoryFactory ff = JmsFactoryFactory.getInstance(JmsConstants.WMQ_PROVIDER);
```

Parametr w wywołaniu metody getInstance() jest stałą, która identyfikuje 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, tak jak przedstawiono to 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. Jeśli aplikacja ma łączyć się w trybie klienckim lub łączyć się z menedżerem kolejek innym niż domyślny menedżer kolejek, przed utworzeniem połączenia aplikacja musi ustawić odpowiednie właściwości obiektu fabryki JmsConnection. Informacje na temat sposobu wykonania tej czynności zawiera sekcja “Ustawianie właściwości obiektów produktu IBM MQ classes for JMS” na stronie 201.

Klasa fabryki JmsFactoryzawiera również metody tworzenia fabryk połączeń dla następujących typów:

- JmsQueueConnectionFactory
- JmsTopicConnectionFactory
- Fabryka JmsXAConnection
- JmsXAQueueConnectionFactory
- JmsXATopicConnectionFactory

Aby utworzyć obiekt kolejki, aplikacja wywołuje metodę createQueue() obiektu fabryki JmsFactory, tak jak przedstawiono to 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 także akceptować identyfikator URI (Uniform Resource Identifier) kolejki jako parametr. Identyfikator URI kolejki to łańcuch, który określa nazwę kolejki produktu IBM MQ i opcjonalnie nazwę menedżera kolejek, który jest właścicielem kolejki, oraz co najmniej jedną właściwość obiektu JmsQueue . Następująca 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 wysłane 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 213. Alternatywny sposób ustawiania właściwości obiektu JmsQueue można znaleźć w sekcji “Ustawianie właściwości obiektów produktu IBM MQ classes for JMS” na stronie 201.

Aby utworzyć obiekt tematu, aplikacja może użyć metody createTopic() obiektu fabryki JmsFactory, tak jak przedstawiono to 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/Football/Results.

Metoda createTopic() może także akceptować identyfikator URI tematu jako parametr. Identyfikator URI tematu to łańcuch, który określa nazwę tematu i opcjonalnie jedną lub więcej właściwości obiektu JmsTopic . Następujące instrukcje zawierają przykładowy identyfikator 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 wysłane do tego miejsca docelowego są nietrwałe i mają priorytet równy 0. Więcej informacji na temat identyfikatorów URI tematów zawiera sekcja “Identyfikatory URI (Uniform Resource Identifier)” na stronie 213. Alternatywny sposób ustawiania właściwości obiektu JmsTopic można znaleźć w sekcji “Ustawianie właściwości obiektów produktu IBM MQ classes for JMS” na stronie 201.

Po utworzeniu fabryki połączeń lub miejsca docelowego przez aplikację ten obiekt może być używany tylko z wybranym dostawcą przesyłania komunikatów.

Ustawianie właściwości obiektów produktu IBM MQ classes for JMS

Aby ustawić właściwości obiektów IBM MQ classes for JMS przy użyciu rozszerzeń IBM JMS , aplikacja korzysta z metod interfejsu com.ibm.msg.client.JmsPropertyContext .

Dla każdego typu danych Java interfejs kontekstu JmsPropertyzawiera metodę ustawiania wartości właściwości z tym typem danych oraz metodę pobierania wartości właściwości z tym typem danych. Na przykład aplikacja wywołuje metodę setIntProperty () w celu ustawienia właściwości o wartości liczby całkowitej i wywołuje metodę getIntProperty () w celu pobrania właściwości z wartością całkowitą.

Instancje klas w pakiecie com.ibm.mq.jms dziedziczą również metody interfejsu kontekstu JmsProperty. Aplikacja może zatem użyć tych metod w celu ustawienia właściwości obiektów MQConnectionFactory, MQQueue i MQTopic.

Gdy aplikacja tworzy obiekt IBM MQ classes for JMS , wszystkie właściwości z wartościami domyślnymi są ustawiane automatycznie. Gdy aplikacja ustawia właściwość, nowa wartość zastępuje poprzednią wartość posiadanej właściwości. Po ustawieniu właściwości nie można jej usunąć, ale jej wartość może zostać zmieniona.

Jeśli aplikacja próbuje ustawić właściwość na wartość, która nie jest poprawną wartością właściwości, produkt IBM MQ classes for JMS zgłasza wyjątek JMSException. Jeśli aplikacja podejmie próbę pobrania właściwości, która nie została ustawiona, zachowanie jest opisane w specyfikacji JMS . IBM MQ classes for JMS zgłasza wyjątek NumberFormatwyjątku dla podstawowych typów danych i zwraca wartość NULL dla przywoływanych typów danych.

Oprócz wstępnie zdefiniowanych właściwości obiektu IBM MQ classes for JMS aplikacja może ustawić własne właściwości. Te zdefiniowane właściwości aplikacji są ignorowane przez produkt IBM MQ classes for JMS.

Więcej informacji na temat właściwości obiektów produktu IBM MQ classes for JMS zawiera sekcja Właściwości obiektów produktu IBM MQ classes for JMS.

Poniższy kod jest przykładem sposobu 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 to, że aplikacja łączy się z menedżerem kolejek QM1 w trybie klienta, używając kanału MQI o nazwie QM1.SVR. Menedżer kolejek jest uruchomiony w systemie o nazwie hosta HOST1, a nasłuchiwanie dla menedżera kolejek nasłuchuje na porcie o numerze 1415. To połączenie i inne połączenia menedżera kolejek powiązane z sesjami, które są pod nią związane, mają powiązaną z nimi 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 (), która może być używana przez aplikację do ustawiania właściwości. Drugi parametr metody to obiekt, który hermetyzuje wartość właściwości. Na przykład następujący kod tworzy obiekt typu Integer, który hermetykuje liczbę całkowitą 1415, a następnie wywołuje właściwość setObject() 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);
```

Z drugiej strony metoda getObjectProperty () zwraca obiekt, który hermetykuje wartość właściwości.

Niejawne przekształcenie wartości właściwości z jednego typu danych na inny.

Jeśli aplikacja używa metody interfejsu kontekstu JmsPropertydo ustawiania lub pobierania właściwości obiektu IBM MQ classes for JMS , wartość tej właściwości może być niejawnie przekształcona z jednego typu danych na inny.

Na przykład poniższa instrukcja ustawia właściwość PRIORITY obiektu JmsQueue q1: .

```
q1.setStringProperty(WMQConstants.WMQ_PRIORITY, "5");
```

Właściwość PRIORITY ma wartość całkowitą, a więc wywołanie właściwości setString() niejawnie przekształca łańcuch "5" (wartość źródłowa) na 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 konwertowana na łańcuch "5" (wartość docelowa) przy użyciu wywołania właściwości getString().

Konwersje obsługiwane przez produkt IBM MQ classes for JMS są wyświetlane w programie Tabela 33 na stronie 203.

Tabela 33. Obsługiwane konwersje z jednego typu danych do innego

Źródłowy typ danych	**Obsługiwane docelowe typy danych**
boolean (boolowskie)	Łańcuch
B	int, long, short, String
char	Łańcuch
double (podwójna)	Łańcuch
liczba zmiennopozycyjna	double, String
int	long, String
long	Łańcuch
short	int, long, String
Łańcuch	boolean, byte, double, float, int, long, short

Ogólne reguły dotyczące obsługiwanych konwersji są następujące:

- Wartości liczbowe mogą być przekształcane z jednego typu danych na inny, pod warunkiem, że w trakcie konwersji nie są tracone 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 na wartość dowolnego innego typu danych (z wyjątkiem `char`) pod warunkiem, że łańcuch jest w poprawnym formacie dla konwersji. Jeśli aplikacja podejmie próbę przekształcenia łańcucha, który nie ma poprawnego formatu, IBM MQ classes for JMS zgłasza wyjątek NumberFormat.
- Jeśli aplikacja podejmie próbę konwersji, która nie jest obsługiwana, produkt IBM MQ classes for JMS zgłasza wyjątek MessageFormat.

Szczegółowe reguły przekształcania wartości z jednego typu danych do drugiego są następujące:

- 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 boolowskim łańcuch "true" (bez rozróżniania wielkości liter) jest przekształcany w łańcuch `true`, a łańcuch "false" (bez rozróżniania wielkości liter) jest przekształcany w wartość `false`. Dowolny inny łańcuch jest przekształcany w `false`.
- Podczas przekształcania łańcucha w wartość o typie danych `byte`, `int`, `long`lub `short`łańcuch musi mieć następujący format:

  [*odstępy*] [*znak*] *cyfry*

  Znaczenia składników tego łańcucha są następujące:

  wartości puste
  Opcjonalne wiodące puste znaki.

sign
Opcjonalny znak plus (+) lub znak minus (-).

cyfry
Ciągła sekwencja cyfr (0-9). Musi istnieć co najmniej jedna cyfra.

Po sekwencji cyfr łańcuch może zawierać inne znaki, które nie są cyframi, ale konwersja zatrzymuje się, gdy tylko pierwszy z tych znaków zostanie osiągnięty. Przyjmuje się, że łańcuch reprezentuje dziesiętną liczbę całkowitą.

Jeśli łańcuch nie ma poprawnego formatu, IBM MQ classes for JMS zgłasza wyjątek NumberFormat.

- Podczas przekształcania łańcucha w wartość typu danych `double` lub `float`łańcuch musi mieć następujący format:

  [*odstępy*] [*znak*] *cyfry* [*e_char* [*znak*] *e_cyfry*]

  Znaczenia składników tego łańcucha są następujące:

  wartości puste
  Opcjonalne wiodące puste znaki.

  sign
  Opcjonalny znak plus (+) lub znak minus (-).

  cyfry
  Ciągła sekwencja cyfr (0-9). Musi istnieć co najmniej jedna cyfra.

  znak
  Znak wykładnika, który ma wartość *E* lub *e*.

  znak
  Opcjonalny znak plus (+) lub znak minus (-) dla wykładnika.

  _cyfry
  Ciągła sekwencja cyfr (0-9) dla wykładnika. Co najmniej jedna cyfra musi być obecna, jeśli łańcuch zawiera znak wykładnika.

  Po sekwencji cyfr lub opcjonalnych znaków reprezentujących wykładnik, łańcuch może zawierać inne znaki, które nie są cyframi, ale konwersja zatrzymuje się, gdy tylko pierwszy z tych znaków zostanie osiągnięty. Przyjmuje się, że łańcuch reprezentuje liczbę dziesiętną zmiennopozycyjną z wykładnikiem, który jest potęgową liczbą 10.

  Jeśli łańcuch nie ma poprawnego formatu, IBM MQ classes for JMS zgłasza wyjątek NumberFormat.

- Podczas przekształcania wartości liczbowej (w tym wartość z typem danych `byte`) do łańcucha, wartość jest przekształcana na łańcuchową reprezentację wartości w postaci liczby dziesiętnej, a nie łańcucha zawierającego znak ASCII dla tej wartości. Na przykład liczba całkowita 65 jest przekształcana w łańcuch "65", a nie na łańcuch "A".

Ustawianie więcej niż jednej właściwości w pojedynczym wywołaniu

Interfejs kontekstu JmsPropertyzawiera również metodę setBatchProperties (), która może być używana przez aplikację do ustawiania więcej niż jednej właściwości w pojedynczym wywołaniu. Parametr metody jest obiektem Map, który hermetyzuje zestaw par nazwa-wartość właściwości.

Na przykład poniższy kod używa metody setBatchProperties () do ustawienia tych samych pięciu właściwości fabryki połączeń, tak jak pokazano to w sekcji “Ustawianie właściwości obiektów produktu IBM MQ classes for JMS” na stronie 201. 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 pamiętać, 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, tak jak pokazano to 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 jest ustawiona.

Nazwy i wartości właściwości

Jeśli aplikacja korzysta z metod interfejsu kontekstu JmsPropertydo ustawiania i pobierania właściwości obiektów produktu IBM MQ classes for JMS , aplikacja może określać nazwy i wartości właściwości w jeden z następujących sposobów. Każdy z dołączonych przykładów pokazuje, jak ustawić właściwość PRIORITY obiektu JmsQueue q1 , tak aby komunikat wysłany do kolejki miał priorytet określony w wywołaniu funkcji send ().

Używanie nazw i wartości właściwości, które są zdefiniowane jako stałe w interfejsie com.ibm.msg.client.wmq.WMQConstants .

Poniższa instrukcja jest przykładem sposobu określania nazw i wartości właściwości w następujący sposób:

```
q1.setIntProperty(WMQConstants.WMQ_PRIORITY, WMQConstants.WMQ_PRI_APP);
```

Korzystanie z nazw i wartości właściwości, które mogą być używane w identyfikatorach URI kolejki i tematu (URI)

Poniższa instrukcja jest przykładem sposobu określania nazw i wartości właściwości w następujący sposób:

```
q1.setIntProperty("priority", -2);
```

W ten sposób można określić tylko nazwy i wartości właściwości miejsc docelowych.

Korzystanie z nazw i wartości właściwości, które są rozpoznawane przez narzędzie administracyjne IBM MQ JMS

Poniższa instrukcja jest przykładem sposobu określania nazw i wartości właściwości w następujący sposób:

```
q1.setStringProperty("PRIORITY", "APP");
```

Skrócona forma nazwy właściwości jest również akceptowalna, co przedstawiono w poniższej instrukcji:

```
q1.setStringProperty("PRI", "APP");
```

Gdy aplikacja pobiera właściwość, zwracana wartość jest zależna od sposobu, w jaki aplikacja określa nazwę właściwości. Na przykład, jeśli aplikacja określa stałą wartość WMQConstants.WMQ_PRIORITY jako nazwę właściwości, zwracana wartość jest liczbą całkowitą -2:

```
int n1 = getIntProperty(WMQConstants.WMQ_PRIORITY);
```

Ta sama wartość jest zwracana, jeśli w aplikacji określono łańcuch "priority" jako nazwę właściwości:

```
int n2 = getIntProperty("priority");
```

Jeśli jednak aplikacja określi łańcuch "PRIORITY" lub "PRI" jako nazwę właściwości, zwracana wartość jest łańcuchem "APP":

```
String s1 = getStringProperty("PRI");
```

Wewnętrznie produkt IBM MQ classes for JMS zapisuje nazwy właściwości i wartości jako wartości literałów zdefiniowane w interfejsie com.ibm.msg.client.wmq.WMQConstants . Jest to zdefiniowany format kanoniczny dla nazw właściwości i wartości. Co do zasady, jeśli aplikacja ustawia właściwości przy użyciu jednego z dwóch innych sposobów określania nazw i wartości właściwości, program IBM MQ classes for JMS musi przekształcić nazwy i wartości z określonego formatu wejściowego w format kanoniczny. Podobnie, jeśli aplikacja pobiera właściwości przy użyciu jednego z dwóch innych sposobów określania nazw i wartości właściwości, program IBM MQ classes for JMS musi przekształcić nazwy z określonego formatu wejściowego w format kanoniczny i przekształcić wartości z formatu kanonicznego w wymagany format wyjściowy. Wykonanie tych konwersji może mieć wpływ na wydajność.

Nazwy właściwości i wartości zwracane przez wyjątki, w plikach śledzenia lub w dzienniku produktu IBM MQ classes for JMS są zawsze w formacie kanonicznym.

Korzystanie z interfejsu mapy

Interfejs kontekstu JmsPropertyrozszerza interfejs java.util.Map . Aplikacja może zatem korzystać z metod interfejsu Map w celu uzyskania dostępu do właściwości obiektu IBM MQ classes for JMS .

Na przykład poniższy kod drukuje nazwy i wartości wszystkich właściwości fabryki połączeń. Kod korzysta tylko z metod interfejsu odwzorowania, aby uzyskać nazwy 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));
}
```

Korzystanie z metod interfejsu Map nie pomija żadnych operacji sprawdzania poprawności właściwości ani konwersji.

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 207.

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 213. Alternatywny sposób ustawiania właściwości obiektu MQQueue można znaleźć w sekcji “Ustawianie właściwości miejsc docelowych” na stronie 208.

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 213. Alternatywny sposób ustawiania właściwości obiektu MQTopic zawiera sekcja “Ustawianie właściwości miejsc docelowych” na stronie 208.

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.

Budowanie połączenia w aplikacji JMS

Aby zbudować połączenie, aplikacja JMS korzysta z obiektu ConnectionFactory w celu utworzenia obiektu połączenia, a następnie uruchamia połączenie.

Aby utworzyć obiekt połączenia, aplikacja korzysta z metody createConnection() obiektu ConnectionFactory , tak jak przedstawiono to w poniższym przykładzie:

```
ConnectionFactory factory;
Connection connection;
.
.
.
connection = factory.createConnection();
```

Gdy tworzone jest połączenie z produktem JMS , IBM MQ classes for JMS tworzy uchwyt połączenia (Hconn) i rozpoczyna konwersację z menedżerem kolejek.

Interfejs fabryczny QueueConnectioni interfejs fabryki TopicConnectiondziedziczą metodę createConnection() z interfejsu ConnectionFactory . W związku z tym można użyć metody createConnection() do utworzenia obiektu specyficznego dla domeny, tak jak pokazano to 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 mającą 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 IllegalState, z 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 z fabryki połączeń specyficznych dla domeny.

Uwaga: Należy zauważyć, że identyfikator procesu aplikacji jest używany jako domyślna tożsamość użytkownika, która ma być przekazywana do menedżera kolejek. Jeśli aplikacja działa w trybie transportu

klienta, ten identyfikator procesu musi istnieć, wraz z odpowiednimi autoryzacjami, na serwerze. Jeśli ma być używana inna tożsamość, należy użyć metody createConnection(nazwa użytkownika, hasło).

Specyfikacja JMS wskazuje, że połączenie jest tworzone w stanie `stopped` . Dopóki połączenie nie zostanie uruchomione, konsument komunikatów, który jest powiązany z połączeniem, nie może odbierać żadnych komunikatów. Aby uruchomić połączenie, aplikacja korzysta z metody start () obiektu Connection, tak jak przedstawiono to w poniższym przykładzie:

```
connection.start();
```

Tworzenie sesji w aplikacji JMS

Aby utworzyć sesję, aplikacja JMS korzysta z metody createSession() obiektu Connection.

Metoda createSession() ma dwa parametry:

1. Parametr, który określa, czy sesja jest transakowana, czy nie,
2. Parametr określający tryb potwierdzania dla sesji

Na przykład następujący kod tworzy sesję, która nie jest transakcjana i ma tryb potwierdzania AUTO_ACKNOWLEDGE:

```
Session session;
.
boolean transacted = false;
session = connection.createSession(transacted, Session.AUTO_ACKNOWLEDGE);
```

Podczas tworzenia sesji programu JMS IBM MQ classes for JMS tworzy uchwyt połączenia (Hconn) i rozpoczyna konwersację z menedżerem kolejek.

Obiekt Session i dowolny obiekt MessageProducer lub MessageConsumer , który został utworzony na podstawie tego obiektu, nie mogą być używane jednocześnie przez różne wątki aplikacji wielowątkowej. Najprostszym sposobem zapewnienia, że obiekty te nie są używane współbieżnie, jest utworzenie osobnego obiektu sesji dla każdego wątku.

Sesje transakcyjne w aplikacjach JMS

Aplikacje produktu JMS mogą uruchamiać transakcje lokalne, najpierw tworząc sesję transakcyjne. Aplikacja może zatwierdzić lub wycofać transakcję.

Aplikacje produktu JMS mogą uruchamiać transakcje lokalne. Transakcja lokalna jest transakcją, która obejmuje zmiany tylko w przypadku zasobów menedżera kolejek, z którym połączona jest aplikacja. Aby uruchomić transakcje lokalne, aplikacja musi najpierw utworzyć sesję transakcyjne, wywołując metodę createSession() obiektu Connection, określając jako parametr, że sesja jest transakcjami transakcjami. Następnie wszystkie komunikaty wysłane i odebrane w ramach sesji są pogrupowane w sekwencję transakcji. Transakcja kończy się, gdy aplikacja zatwierdza lub wycofuje komunikaty, które zostały wysłane i odebrane od momentu rozpoczęcia transakcji.

Aby zatwierdzić transakcję, aplikacja wywołuje metodę commit () obiektu Session. Jeśli transakcja zostanie zatwierdzona, wszystkie komunikaty wysłane w ramach transakcji stają się dostępne do dostarczenia do innych aplikacji, a wszystkie komunikaty odebrane w ramach transakcji zostaną potwierdzone, tak 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 odrzucane przez serwer przesyłania komunikatów, a wszystkie komunikaty odebrane w ramach transakcji stają się dostępne do dostarczenia ponownie. W domenie typu punkt z punktem komunikaty, które zostały odebrane, są ponownie umieszczane w ich kolejkach i ponownie stają się widoczne dla innych aplikacji.

Nowa transakcja jest uruchamiana automatycznie, gdy aplikacja tworzy sesję transakcyjna lub wywołuje metodę commit () lub rollback (). Oznacza to, że sesja transakcyjna zawsze ma aktywną transakcję.

Gdy aplikacja zamknie sesję transakcyjne, następuje niejawne wycofanie zmian. Jeśli aplikacja zamknie połączenie, dla wszystkich sesji transakcyjnych połączenia zostanie wykonane niejawne wycofanie zmian.

Jeśli aplikacja zakończy działanie bez zamykania połączenia, dla wszystkich sesji transakcyjnych połączenia zostanie również wykonane niejawne wycofanie zmian.

Transakcja jest w całości zawarta w ramach sesji transakcyjnej. Transakcja nie może obejmować sesji. Oznacza to, że aplikacja nie może wysyłać i odbierać komunikatów w dwóch lub większej liczby sesji transakcyjnych, a następnie zatwierdzać lub wycofywanie wszystkich tych działań jako jednej transakcji.

Tryby potwierdzania sesji programu JMS
Każda sesja, która nie jest transakcowana, ma tryb potwierdzenia, który określa sposób, w jaki odbierane są komunikaty odbierane przez aplikację. Dostępne są trzy tryby potwierdzenia, a wybór trybu potwierdzania wpływa na konstrukcję aplikacji.

Jeśli sesja nie jest transakowana, sposób potwierdzania komunikatów odbieranych przez aplikację jest określany przez tryb potwierdzania sesji. Trzy tryby potwierdzenia są opisane w następujących akapitach:

AUTO_POTWIERDZANIE

Sesja automatycznie potwierdzi każdy komunikat otrzymany przez aplikację.

Jeśli komunikaty są dostarczane synchronicznie do aplikacji, sesja potwierdza otrzymanie komunikatu za każdym razem, gdy wywołanie Odbiór zakończy się pomyślnie. Jeśli komunikaty są dostarczane asynchronicznie, sesja potwierdza otrzymanie komunikatu za każdym razem, gdy wywołanie metody onMessage() procesu nasłuchiwania komunikatów zakończy się pomyślnie.

Jeśli aplikacja pomyślnie odbierze komunikat, ale niepowodzenie uniemożliwia potwierdzenie wystąpienia, komunikat staje się dostępny do ponownego dostarczenia. Dlatego aplikacja musi być w stanie obsłużyć komunikat, który został ponownie dostarczony.

DUPS_OK_ACKNOWLEDGE

Sesja potwierdza komunikaty odebrane przez aplikację w czasie, gdy jest ona wybierana.

Użycie tego trybu potwierdzenia zmniejsza ilość pracy, jaką musi wykonać sesja, ale błąd, który uniemożliwia potwierdzenie komunikatu, może spowodować ponowne udostępnienie więcej niż jednego komunikatu do dostarczenia. W związku z tym aplikacja musi mieć możliwość obsługi komunikatów, które są ponownie dostarczane.

Ograniczenie: W trybach AUTO_ACKNOWLEDGE i DUPS_OK_ACKNOWLEDGE program JMS nie obsługuje aplikacji zgłaszających nieobsługiwany wyjątek w obiekcie nasłuchiwania komunikatów. Oznacza to, że komunikaty są zawsze potwierdzane w momencie powrotu programu nasłuchującego komunikatów, niezależnie od tego, czy został on pomyślnie przetworzony (pod warunkiem, że wszystkie niepowodzenia są niekrytyczne i nie uniemożliwiają kontynuowania aplikacji). Jeśli wymagane jest dokładniejsze sterowanie potwierdzeniem komunikatów, należy użyć trybów CLIENT_ACKNOWLEDGE lub transacted, które nadają aplikacji pełną kontrolę nad funkcjami potwierdzania.

POTWIERDZANIE_KLIENTA

Aplikacja potwierdza otrzymywane przez niego komunikaty, wywołując metodę Acknowledge klasy Message.

Aplikacja może potwierdzić odbiór każdego komunikatu indywidualnie lub może otrzymać partię komunikatów i wywołać metodę Acknowledge tylko dla ostatniego komunikatu, który otrzymuje. Gdy metoda Acknowledge jest nazywana wszystkimi wiadomościami otrzymanymi od czasu ostatniego wywołania metody, są one potwierdzane.

W połączeniu z dowolnym z tych trybów potwierdzania aplikacja może zatrzymać i ponownie uruchomić dostarczanie komunikatów w sesji, wywołując metodę Recover klasy Session. Komunikaty odebrane, ale wcześniej niepotwierdzone, są ponownie dostarczane. Mogą one jednak nie być dostarczane w tej samej kolejności, w jakiej zostały dostarczone wcześniej. W międzyczasie mogły zostać wysłane komunikaty o wyższym priorytecie, a niektóre z oryginalnych komunikatów mogły utraci ważność. W domenie typu punkt z punktem niektóre z oryginalnych komunikatów mogły zostać skonsumowane przez inną aplikację.

Aplikacja może określić, czy komunikat jest ponownie dostarczany, sprawdzając zawartość pola nagłówka JMSRedelivered komunikatu. Aplikacja wykonuje tę funkcję, wywołując 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 213.

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 213.

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. Więcej informacji na ten temat zawiera sekcja Używanie łańcuchów tematów.

***propertyName1*, *propertyName2*, ...**
Nazwy właściwości obiektu kolejki lub tematu utworzonych przez aplikację. Tabela 34 na stronie 214 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 34 na stronie 214 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 34 na stronie 214 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 34. 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.

Tabela 34. Nazwy właściwości i poprawne wartości do użycia w identyfikatorach URI kolejek i tematów (kontynuacja)

Nazwa właściwości	**Opis**	**Poprawne wartości**
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 .
Utrata 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. • 0-komunikat wysłany do miejsca docelowego nigdy nie traci ważności. • Dodatnia liczba całkowita określająca czas życia w milisekundach.
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. • -1-ASCF (system kaskadowy) • 0 - wyłączone • 3-NOTR • 5-NIEZAWODNY • 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. • -1-zgodnie z określeniem w atrybucie *DefPersistence* kolejki lub tematu IBM MQ . • 1-Nietrwały. • 2-Trwałe. • 3-odpowiednik wartości HIGH właściwości PERSISTENCE używanej w narzędziu administracyjnym IBM MQ JMS . Wyjaśnienie tej wartości zawiera sekcja “Komunikaty trwałe produktu JMS” na stronie 241.

Tabela 34. Nazwy właściwości i poprawne wartości do użycia w identyfikatorach URI kolejek i tematów (kontynuacja)

Nazwa właściwości	Opis	Poprawne wartości
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. • -1-zgodnie z określeniem w atrybucie *DefPriority* kolejki lub tematu IBM MQ . • 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 . • 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 może 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 korzysta z obiektu MessageProducer do wysyłania komunikatów. Aplikacja zwykle tworzy obiekt MessageProducer dla określonego miejsca docelowego, które może być kolejką lub tematem, tak aby wszystkie komunikaty wysłane przy użyciu producenta komunikatów były wysyłane do tego samego miejsca docelowego. Dlatego zanim aplikacja może utworzyć obiekt MessageProducer , musi on najpierw utworzyć obiekt kolejki lub tematu. Informacje na temat tworzenia kolejki lub obiektu tematu zawierają następujące tematy:

- “Korzystanie z interfejsu JNDI do pobierania obiektów administrowanych w aplikacji JMS” na stronie 198
- “Korzystanie z rozszerzeń IBM JMS” na stronie 199

- “Korzystanie z rozszerzeń IBM MQ JMS” na stronie 206
- “Tworzenie miejsc docelowych w aplikacji JMS” na stronie 212

Aby utworzyć obiekt MessageProducer , aplikacja korzysta z metody createProducer() obiektu Session, tak jak przedstawiono to w poniższym przykładzie:

```
MessageProducer producer = session.createProducer(destination);
```

Parametr `destination` to obiekt kolejki lub tematu, który został wcześniej utworzony przez aplikację.

Zanim aplikacja może wysłać komunikat, musi utworzyć obiekt komunikatu. Treść komunikatu zawiera dane aplikacji, a program JMS definiuje pięć typów treści komunikatu:

- Bajty
- Odwzoruj
- Obiekt
- Strumień
- Tekstowy

Każdy typ treści komunikatu ma własny interfejs JMS , który jest podinterfejsem interfejsu komunikatu, a także metodę w interfejsie Session w celu utworzenia komunikatu z tym typem treści. Na przykład interfejs dla komunikatu tekstowego ma nazwę TextMessage, a aplikacja korzysta z metody createTextMessage () obiektu Session w celu utworzenia komunikatu tekstowego, tak jak przedstawiono to w następującej instrukcji:

```
TextMessage outMessage = session.createTextMessage(outString);
```

Więcej informacji na temat komunikatów i treści komunikatów zawiera sekcja “Komunikaty produktu JMS” na stronie 138.

Aby wysłać komunikat, aplikacja używa metody send () obiektu MessageProducer , tak jak przedstawiono to 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 TopicPublisher, podinterfejs MessageProducer , który jest specyficzny dla domeny publikowania/subskrybowania, ma również metodę publikowania (), która może być używana zamiast metody send (). Te dwie metody są funkcjonalnie takie same.

Aplikacja może utworzyć obiekt MessageProducer bez określonego miejsca docelowego. W takim przypadku aplikacja musi określić miejsce docelowe podczas wywoływania metody send ().

Jeśli aplikacja wysyła komunikat w ramach transakcji, komunikat nie zostanie dostarczony do miejsca docelowego, dopóki transakcja nie zostanie zatwierdzona. Oznacza to, że aplikacja nie może wysłać komunikatu i otrzymać odpowiedzi na komunikat w ramach tej samej transakcji.

Miejsce docelowe można skonfigurować w taki sposób, aby gdy aplikacja wysyła do niego komunikaty, program IBM MQ classes for JMS przekazuje komunikat i zwraca kontrolę z powrotem do aplikacji bez określania, czy menedżer kolejek odebrał komunikat w sposób bezpieczny. Jest to czasami nazywane *umieszczonym asynchronicznie*. Więcej informacji na ten temat zawiera sekcja “Asynchronicznie umieszczanie komunikatów w produkcie IBM MQ classes for JMS” na stronie 316.

Odbieranie komunikatów w aplikacji JMS

Aplikacja używa konsumenta komunikatów do odbierania komunikatów. Trwałym subskrybentem tematu jest konsument komunikatów, który odbiera wszystkie komunikaty wysłane do miejsca docelowego, włącznie z tymi, które są wysyłane, gdy konsument jest nieaktywny. Aplikacja może wybrać, które

komunikaty mają być odbierane za pomocą selektora komunikatów, i może odbierać komunikaty asynchronicznie, korzystając z obiektu nasłuchiwania komunikatów.

Aplikacja korzysta z obiektu MessageConsumer do odbierania komunikatów. Aplikacja tworzy obiekt MessageConsumer dla określonego miejsca docelowego, które może być kolejką lub tematem, dzięki czemu wszystkie komunikaty odebrane z użyciem konsumenta komunikatów są odbierane z tego samego miejsca docelowego. Dlatego zanim aplikacja może utworzyć obiekt MessageConsumer , musi on najpierw utworzyć obiekt kolejki lub tematu. Informacje na temat tworzenia kolejki lub obiektu tematu zawierają następujące tematy:

- “Korzystanie z interfejsu JNDI do pobierania obiektów administrowanych w aplikacji JMS” na stronie 198
- “Korzystanie z rozszerzeń IBM JMS” na stronie 199
- “Korzystanie z rozszerzeń IBM MQ JMS” na stronie 206
- “Tworzenie miejsc docelowych w aplikacji JMS” na stronie 212

Aby utworzyć obiekt MessageConsumer , aplikacja korzysta z metody createConsumer() obiektu Session, tak jak przedstawiono to w poniższym przykładzie:

```
MessageConsumer consumer = session.createConsumer(destination);
```

Parametr `destination` to obiekt kolejki lub tematu, który został wcześniej utworzony przez aplikację.

Następnie aplikacja korzysta z metody receive () obiektu MessageConsumer w celu odebrania komunikatu z miejsca docelowego, jak pokazano w poniższym przykładzie:

```
Message inMessage = consumer.receive(1000);
```

Parametr w wywołaniu metody receive () określa czas (w milisekundach), przez jaki metoda oczekuje na dotarcie odpowiedniego komunikatu, jeśli komunikat nie jest dostępny natychmiast. Jeśli ten parametr zostanie pominięty, połączenia będą blokowe przez czas nieokreślony aż do momentu nadejścia odpowiedniego komunikatu. Jeśli nie chcesz, aby aplikacja oczekiwała na komunikat, użyj metody receiveNoWait ().

Metoda receive () zwraca komunikat o określonym typie. Na przykład, gdy aplikacja odbierze komunikat tekstowy, obiekt zwrócony przez wywołanie receive () jest obiektem typu TextMessage .

Jednak zadeklarowany typ obiektu zwrócony przez wywołanie receive () jest obiektem komunikatu. Dlatego w celu wyodrębnienia danych z treści komunikatu, który został właśnie odebrany, aplikacja musi rzutować z klasy Message na bardziej konkretną podklasę, taką jak TextMessage. Jeśli typ komunikatu nie jest znany, aplikacja może użyć operatora `instanceof` w celu określenia typu. Zawsze zaleca się stosowanie aplikacji w celu określenia typu komunikatu przed rzutowaniem, dzięki czemu błędy mogą być obsługiwane w sposób wdzięczny.

Poniższy kod używa operatora `instanceof` i pokazuje, jak wyodrębnić dane z treści wiadomości tekstowej:

```
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 zostanie dostarczony do miejsca docelowego, dopóki transakcja nie zostanie zatwierdzona. Oznacza to, że aplikacja nie może wysłać komunikatu i otrzymać odpowiedzi na komunikat w ramach tej samej transakcji.

Jeśli konsument komunikatów odbierze 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/subskrybowania produkt JMS identyfikuje dwa typy konsumenta komunikatów, nietrwały subskrybent tematu i trwały subskrybenta tematów, które są opisane w następujących dwóch sekcjach.

Nietrwałe subskrybenty tematów

Nietrwały subskrybent tematu otrzymuje tylko te komunikaty, które są publikowane, gdy subskrybent jest aktywny. Subskrypcja nietrwała jest uruchamiana, gdy aplikacja tworzy nietrwały subskrybent tematu i kończy się, gdy aplikacja zamknie subskrybenta, lub gdy subskrybent nie wchodzi w zakres. Jako rozszerzenie w produkcie IBM MQ classes for JMSsubskrybent tematu nietrwałego otrzymuje również zachowane publikacje.

Aby utworzyć nietrwały subskrybent tematów, aplikacja może użyć niezależnej metody createConsumer() domeny, określając obiekt tematu jako miejsce docelowe. Alternatywnie aplikacja może użyć metody createSubscriber() specyficznej dla domeny, tak jak przedstawiono to w poniższym przykładzie:

```
TopicSubscriber subscriber = session.createSubscriber(topic);
```

Parametr `topic` jest obiektem tematu, który został wcześniej utworzony przez aplikację.

Trwałe subskrybenty tematów

Ograniczenie: Aplikacja nie może utworzyć trwałych subskrybentów tematów podczas korzystania z połączenia w czasie rzeczywistym z brokerem.

Trwały subskrybent tematu odbiera wszystkie komunikaty opublikowane w czasie trwania trwałej subskrypcji. Komunikaty te obejmują wszystkie te komunikaty, które są publikowane, gdy subskrybent nie jest aktywny. Jako rozszerzenie w produkcie IBM MQ classes for JMS, trwały subskrybent tematu otrzymuje również zachowane publikacje.

Aby utworzyć trwały subskrybent tematów, aplikacja korzysta z metody createDurableSubscriber () obiektu Session, tak jak przedstawiono to w poniższym przykładzie:

```
TopicSubscriber subscriber = session.createDurableSubscriber(topic, "D_SUB_000001");
```

W wywołaniu metody createDurableSubscriber () pierwszy parametr jest obiektem tematu, który został wcześniej utworzony przez aplikację, a drugim parametrem jest nazwa używana do identyfikowania trwałej subskrypcji.

Sesja użyta do utworzenia 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, a zatem identyfikator klienta stanowi część pełnego, unikalnego identyfikatora trwałej subskrypcji. Aby kontynuować korzystanie z trwałej subskrypcji, która została wcześniej utworzona, aplikacja musi utworzyć trwałą subskrybent tematu przy użyciu sesji o takim samym identyfikatorze klienta, jak ten powiązany z trwałą subskrypcją, i przy użyciu tej samej nazwy subskrypcji.

Trwała subskrypcja jest uruchamiana, gdy aplikacja tworzy trwały subskrybent tematu przy użyciu identyfikatora klienta i nazwy subskrypcji, dla której nie istnieje obecnie trwała subskrypcja. Jednak trwała subskrypcja nie kończy się, gdy aplikacja zamknie trwałą subskrybent tematu. Aby zakończyć trwałą subskrypcję, aplikacja musi wywołać metodę unsubscribe () obiektu Session, który ma taki sam identyfikator klienta, jak ten powiązany z trwałą subskrypcją. Parametrem w wywołaniu unsubscribe () jest nazwa subskrypcji, tak jak przedstawiono to w poniższym przykładzie:

```
session.unsubscribe("D_SUB_000001");
```

Zasięgiem trwałej subskrypcji jest menedżer kolejek. Jeśli w jednym menedżerze kolejek istnieje trwała subskrypcja, a aplikacja połączona z innym menedżerem kolejek tworzy trwałą subskrypcję z tym samym identyfikatorem klienta i nazwą subskrypcji, to dwie trwałe subskrypcje są całkowicie niezależne.

Selektory komunikatów

Aplikacja może określić, że tylko te komunikaty, które spełniają określone kryteria, są zwracane przez kolejne wywołania receive (). Podczas tworzenia obiektu MessageConsumer aplikacja może określić wyrażenie SQL (Structured Query Language) określające, które komunikaty są pobierane. To wyrażenie SQL jest nazywane *selektorem komunikatów*. Selektor komunikatów może zawierać nazwy pól nagłówka komunikatu produktu JMS oraz właściwości komunikatu. Informacje na temat konstruowania selektora komunikatów zawiera sekcja “Selektory komunikatów w produkcie JMS” na stronie 138.

W poniższym przykładzie przedstawiono, w jaki sposób aplikacja może wybierać 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 w stanie życia tego konsumenta. Jeśli aplikacja wymaga więcej niż jednego selektora komunikatów, aplikacja musi utworzyć konsumenta komunikatów dla każdego selektora komunikatów.

Należy zauważyć, że gdy aplikacja jest połączona z menedżerem kolejek produktu IBM WebSphere MQ 7 , właściwość MSGSELECTION fabryki połączeń nie ma wpływu. Aby zoptymalizować wydajność, wszystkie opcje wyboru komunikatów są wykonywane przez menedżer kolejek.

Pomijanie publikacji lokalnych

Aplikacja może utworzyć konsument komunikatów, który ignoruje publikacje opublikowane w ramach połączenia własnego konsumenta. Aplikacja wykonuje tę operację, ustawiając trzeci parametr w wywołaniu metody createConsumer() na wartość `true`, jak pokazano w poniższym przykładzie:

```
MessageConsumer consumer = session.createConsumer(topic, null, true);
```

W wywołaniu metody createDurableSubscriber () aplikacja wykonuje tę funkcję, ustawiając czwarty parametr na wartość `true`, jak to 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 za pomocą konsumenta komunikatów. Program nasłuchujący komunikatów ma metodę o nazwie onMessage, która jest wywoływana asynchronicznie, gdy dostępny jest odpowiedni komunikat i którego celem jest przetworzenie komunikatu. Poniższy kod ilustruje mechanizm:

```
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 w celu synchronicznego odbierania komunikatów za pomocą wywołań receive () lub asynchronicznego odbierania komunikatów za pomocą programów nasłuchujących komunikatów, ale nie dla obu tych elementów. Jeśli aplikacja musi odbierać komunikaty synchronicznie i asynchronicznie, musi utworzyć oddzielne sesje.

Po ustawieniu sesji w celu asynchronicznego odbierania komunikatów nie można wywoływać następujących metod w tej sesji ani w obiektach utworzonych w tej sesji:

- MessageConsumer.receive ()
- MessageConsumer.receive (long)
- MessageConsumer.receiveNoWait ()
- Session.acknowledge()
- MessageProducer.send (miejsce docelowe, komunikat)
- MessageProducer.send (Destination, Message, int, int, long)
- MessageProducer.send (Message)
- MessageProducer.send (Message, int, int, long)
- MessageProducer.send (miejsce docelowe, komunikat, obiekt CompletionListener)
- MessageProducer.send (Destination, Message, int, int, long, CompletionListener)
- MessageProducer.send (Message, 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(Destination, String, boolean).
- Session.createDurableSubscriber(Topic, String)
- Session.createDurableSubscriber(Topic, String, String, boolean)
- Session.createMapMessage()
- Session.createMessage()
- Session.createObjectMessage()
- Session.createObjectMessage(Serializable)
- Session.createProducer(Miejsce docelowe)
- Session.createQueue(String).

- Session.createStreamMessage()
- Session.createTemporaryQueue()
- Session.createTemporaryTopic()
- Session.createTextMessage()
- Session.createTextMessage(String)
- Session.createTopic()
- Session.getAcknowledgeMode()
- Session.getMessageListener()
- Session.getTransacted()
- Session.rollback()
- Session.unsubscribe(String).

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'
```

.

Odbieranie komunikatów trujących

Aplikacja może otrzymać komunikat, który nie może zostać przetworzony. Może istnieć kilka przyczyn, dla których komunikat nie może zostać przetworzony, na przykład komunikat może mieć niepoprawny format. Takie komunikaty są opisywane jako nieprzetwarzalne komunikaty i wymagają specjalnej obsługi, aby zapobiec rekurencyjnemu przetwarzaniu komunikatu.

Szczegółowe informacje na temat obsługi nieprzetwarzalnych komunikatów zawiera sekcja “Obsługa komunikatów nieprzetwarzalnych w produkcie IBM MQ classes for JMS” na stronie 223.

Pobieranie danych użytkownika subskrypcji

Jeśli komunikaty, które aplikacja IBM MQ classes for JMS konsumuje z kolejki, są umieszczane przez zdefiniowaną administracyjnie trwałą subskrypcję, aplikacja musi uzyskać dostęp do informacji o danych użytkownika powiązanych z subskrypcją. Te informacje są dodawane do komunikatu jako właściwość.

Jeśli komunikat jest pobierany z kolejki zawierającej nagłówek RFH2 z folderem MQPS, wartość powiązana z kluczem Sud, o ile istnieje, jest dodawana jako właściwość String do obiektu komunikatu produktu JMS zwracanego do aplikacji IBM MQ classes for JMS . Aby włączyć pobieranie tej właściwości z komunikatu, należy użyć stałej JMS_IBM_SUBSCRIPTION_USER_DATA w interfejsie JmsConstants z metodą javax.jms.Message.getStringProperty(java.lang.String), aby pobrać dane użytkownika subskrypcji.

W poniższym przykładzie subskrypcja trwała 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 trwałą subskrypcją są następnie dodawane jako właściwość do wiadomości, która jest przechowywana w folderze MQPS nagłówka RFH2 z kluczem Sud.

Aplikacja IBM MQ classes for JMS może wywołać następujące informacje:

```
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 143

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 zamykała niektóre obiekty JMS jawnie przed zatrzymaniem. Nie można wywoływać procesów finalizujących, dlatego nie należy ich używać do wolnych zasobów. Nie zezwalaj na zakończenie działania aplikacji ze skompresowanym śledzeniem.

Samo czyszczenie pamięci nie może w odpowiednim czasie zwolnić wszystkich zasobów IBM MQ classes for JMS i IBM MQ , zwłaszcza jeśli aplikacja tworzy wiele krótkotrych obiektów JMS na poziomie sesji lub niższych. Dlatego ważne jest, aby aplikacja zamykała obiekt Connection, Session, MessageConsumerlub obiekt MessageProducer , gdy nie jest już wymagany.

Jeśli aplikacja zakończy działanie bez zamykania połączenia, dla wszystkich sesji transaktywnych połączenia zostanie wykonane niejawne wycofanie zmian. Aby zapewnić, że wszystkie zmiany wprowadzone przez aplikację zostaną zatwierdzone, należy zamknąć połączenie jawnie przed zamknięciem aplikacji.

Nie należy używać finalizatorów w aplikacji do zamykania obiektów produktu JMS . Ponieważ nie można wywołać procesu finalizowania, zasoby mogą nie zostać zwolnione. Po zamknięciu połączenia zamyka wszystkie sesje utworzone na jego podstawie. Podobnie elementy MessageConsumers i MessageProducers utworzone na podstawie sesji są zamykane po zamknięciu sesji. Jednak należy jawnie rozważyć zamknięcie sesji, MessageConsumersi MessageProducers w celu zapewnienia, że zasoby są zwalniane w odpowiednim czasie.

Jeśli kompresja śledzenia jest aktywowana, wyłączenia System.Halt() i nieprawidłowe, niekontrolowane zakończenie działania maszyny JVM prawdopodobnie spowodują uszkodzenie pliku śledzenia. Jeśli to możliwe, wyłącz narzędzie śledzenia, gdy zebrano potrzebne informacje śledzenia. W przypadku śledzenia aplikacji aż do nieprawidłowego zakończenia należy użyć nieskompresowanych danych wyjściowych śledzenia.

Uwaga: Aby rozłączyć się z menedżerem kolejek, aplikacja JMS wywołuje metodę close () w obiekcie 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 for IBM MQ 9.1.0 Fix Pack 1 i wcześniejszych wersji, lub IBM MQ 9.1.1, jeśli kolejka lokalna lub kolejka docelowa dla kolejki aliasowej jest kolejką klastra, należy otworzyć kolejkę IBM MQ classes for JMS 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.

- V 9.1.0.2 V 9.1.2 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 332
W obiektach serwera aplikacji obsługa komunikatów nieprzetwarzalnych jest nieco inna niż w innych miejscach w produkcie IBM MQ classes for JMS.

Wyjątki w produkcie IBM MQ classes for JMS

Aplikacja IBM MQ classes for JMS musi obsługiwać wyjątki, które są zgłaszane przez wywołania funkcji API produktu JMS lub dostarczane do procedury obsługi wyjątków.

Produkt IBM MQ classes for JMS zgłasza problemy w czasie wykonywania, zgłaszając wyjątki. Wyjątek JMSException jest klasą główną dla wyjątków zgłaszanych przez metody produktu JMS , a wychwytywanie wyjątków JMSException udostępnia ogólny sposób obsługi wszystkich wyjątków związanych z produktem JMS .

Każdy wyjątek JMSException hermetyzuje następujące informacje:

- Komunikat o wyjątku specyficznym dla dostawcy, który jest wyświetlany przez aplikację wywołując metodę Throwable.getMessage().
- Kod błędu specyficznego dla dostawcy, który jest pobierany przez wywołanie metody JMSException.getErrorCode().
- Powiązany wyjątek. Wyjątek zgłoszony przez wywołanie funkcji API produktu JMS często jest wynikiem problemu niższego poziomu, który jest zgłaszany przez inny wyjątek powiązany z tym wyjątkiem. Aplikacja uzyskuje powiązany wyjątek, wywołując metodę JMSException.getLinkedException() lub metodę Throwable.getCause().

Większość wyjątków zgłaszanych przez produkt IBM MQ classes for JMS to instancje podklas wyjątku JMSException. Te podklasy implementują interfejs com.ibm.msg.client.jms.JmsExceptionDetail , który udostępnia następujące dodatkowe informacje:

- Wyjaśnienie komunikatu o wyjątku, którego dotyczy aplikacja, wywołując metodę JmsExceptionDetail.getExplanation().
- Zalecana odpowiedź użytkownika na wyjątek, który aplikacja uzyskuje, wywołując metodę JmsExceptionDetail.getUserAction().
- Klucze dla operacji wstawiania komunikatu w komunikacie o wyjątku. Aplikacja uzyskuje iterator dla wszystkich kluczy, wywołując metodę JmsExceptionDetail.getKeys().
- Komunikat zostanie wstawiony w komunikacie o wyjątku. Na przykład wstawienie komunikatu może być nazwą kolejki, która spowodowała wyjątek, i może być przydatna dla aplikacji, która ma mieć dostęp do tej nazwy. Aplikacja uzyskuje wstawiany komunikat odpowiadający określonowi kluczowi, wywołując metodę JmsExceptionDetail.getValue().

Wszystkie metody w interfejsie szczegółów JmsExceptionmogą zwracać 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.
```

Zgłoszony wyjątek, com.ibm.msg.client.jms.DetailedInvalidDestinationException, jest podklasą klasy javax.jms.InvalidDestinationException i implementuje interfejs com.ibm.msg.client.jms.JmsExceptionDetail .

Powiązane wyjątki

Dołączony wyjątek zawiera dodatkowe informacje na temat problemu z środowiskiem wykonawczym. Dlatego dla każdego zgłoszonego wyjątku JMSException aplikacja powinna sprawdzić powiązany wyjątek. Sam powiązany wyjątek może mieć inny powiązany wyjątek, a więc połączone wyjątki tworzą łańcuch prowadzący do pierwotnego problemu bazowego. Powiązany wyjątek jest implementowany przy użyciu mechanizmu połączonych wyjątków klasy java.lang.Throwable , a aplikacja uzyskuje

powiązany wyjątek, wywołując metodę Throwable.getCause(). W przypadku wyjątku JMSException metoda getLinkedException () w rzeczywistości deleguje do metody Throwable.getCause().

Na przykład, jeśli aplikacja określa niepoprawny numer portu podczas nawiązywania połączenia z menedżerem kolejek, wyjątki tworzą następujący łańcuch:

```
com.ibm.msg.client.jms.DetailIllegalStateException
|
+- -->
com.ibm.mq.MQException
|
+- -->
com.ibm.mq.jmqi.JmqiException
|
+- -->
java.net.ConnectionException
```

Zwykle każdy wyjątek w łańcuchu jest zgłaszany z innej warstwy w kodzie. Na przykład wyjątki w poprzedzającym łańcuchu są zgłaszane przez następujące warstwy:

- Pierwszy wyjątek, instancja podklasy wyjątku JMSException, jest zgłaszany przez wspólną warstwę w produkcie IBM MQ classes for JMS.
- Następny wyjątek, instancja klasy com.ibm.mq.MQException, jest zgłaszany przez dostawcę przesyłania komunikatów produktu IBM MQ .
- Następny wyjątek, instancja klasy com.ibm.mq.jmqi.JmqiException, jest zgłaszany przez wspólny interfejs produktu Java do interfejsu MQI.
- Końcowy wyjątek, instancja klasy java.net.ConnectionException, jest zgłaszany przez bibliotekę klas Java .

Więcej informacji na temat architektury warstwowej produktu IBM MQ classes for JMSmożna znaleźć w sekcji Klasy produktuIBM MQ dla architektury JMS.

Używając kodu podobnego do następującego kodu, aplikacja może iterować przez ten łańcuch w celu wyodrębnienia wszystkich odpowiednich informacji:

```
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 pamiętać, że aplikacja powinna zawsze sprawdzać typ każdego wyjątku w łańcuchu, ponieważ typ wyjątku może się różnić, a wyjątki różnych typów hermetyzować różne informacje.

Uzyskiwanie informacji specyficznych dla produktu IBM MQ dotyczących problemu

Instancje com.ibm.mq.MQException i com.ibm.mq.jmqi.JmqiException hermetyzują IBM MQ konkretne informacje o problemie.

Wyjątek MQException hermetyzuje następujące informacje:

- Kod zakończenia, który aplikacja uzyskuje, wywołując metodę getCompCode ().
- Kod przyczyny, który aplikacja uzyskuje przez wywołanie metody getReason().

Wyjątek JmqiException hermetyzuje kod zakończenia i kod przyczyny. Dodatkowo jednak wyjątek JmqiException hermetykuje informacje w komunikacie AMQ *nnnn* lub CSQ *nnnn* , jeśli jest on powiązany z tym wyjątkiem. Wywołując odpowiednie metody wyjątku, aplikacja może uzyskać różne komponenty tego komunikatu, takie jak istotność, wyjaśnienie i odpowiedź użytkownika.

Przykłady użycia metod wymienionych w tej sekcji można znaleźć w przykładowym kodzie w produkcie “Powiązane wyjątki” na stronie 227.

Aktualizacja z poprzednich wersji produktu IBM MQ classes for JMS

W porównaniu z poprzednimi wersjami produktu IBM MQ classes for JMSwiększość kodów błędów i komunikatów o wyjątkach została zmieniona w IBM WebSphere MQ 7.0. Przyczyną tych zmian jest fakt, że w produkcie IBM WebSphere MQ 7.0produkt IBM MQ classes for JMS ma architekturę warstwową, a wyjątki są zgłaszane z różnych warstw w kodzie.

Jeśli na przykład aplikacja próbuje połączyć się z menedżerem kolejek, który nie istnieje, poprzednia wersja produktu IBM MQ classes for JMS zgłosiła wyjątek JMSException z następującymi informacjami:

```
MQJMS2005: Failed to create MQQueueManager for 'localhost:QM_test'.
```

Ten wyjątek zawierał powiązany wyjątek MQException z następującymi informacjami:

```
MQJE001: Completion Code 2, Reason 2058
```

W porównaniu w tych samych okolicznościach wersja 7.0 produktu IBM MQ classes for JMS zgłasza wyjątek JMSException z następującymi informacjami:

```
Message : JMSWMQ0018: Failed to connect to queue manager 'QM_test' with
connection mode 'Client' and host name 'localhost'.
Class : class com.ibm.msg.client.jms.DetailedJMSException
Error Code : JMSWMQ0018
Explanation : null
User Action : Check the queue manager is started and if running in client mode,
check there is a listener running. Please see the linked exception
for more information.
```

Ten wyjątek zawiera powiązany wyjątek MQException z następującymi informacjami:

```
Message : JMSCMQ0001: IBM MQ call failed with compcode '2' ('MQCC_FAILED')
          reason '2058' ('MQRC_Q_MGR_NAME_ERROR').
```

```
Class : class com.ibm.mq.MQException
Completion Code : 2
Reason Code : 2058
```

Jeśli aplikacja analizuje lub testuje komunikaty wyjątków zwracane przez metodę Throwable.getMessage() lub kody błędów zwracane przez metodę JMSException.getErrorCode(), a aktualizacja jest wykonywana z wersji wcześniejszej niż IBM WebSphere MQ 7.0, aplikacja prawdopodobnie musi zostać zmodyfikowana w celu użycia wersji 7.0 lub nowszej produktu IBM MQ classes for JMS.

Obiekty nasłuchiwania wyjątków

Aplikacja może zarejestrować obiekt nasłuchiwania wyjątków w obiekcie połączenia. Następnie, jeśli wystąpi problem, który sprawia, że połączenie nie będzie możliwe do użycia, produkt IBM MQ classes for JMS dostarczy wyjątek do obiektu nasłuchiwania wyjątków, wywołując metodę onException(). Następnie aplikacja ma możliwość ponownego nawiązania połączenia. Produkt IBM MQ classes for JMS może również dostarczyć wyjątek do obiektu nasłuchiwania wyjątków, jeśli wystąpi problem podczas asynchronicznego dostarczania komunikatu.

W produkcie IBM MQ 8.0.0 Fix Pack 2, aby zachować zachowanie dla bieżących aplikacji JMS, które konfigurują obiekt JMS MessageListener i JMS ExceptionListener, oraz aby upewnić się, że produkt IBM MQ classes for JMS jest spójny ze specyfikacją JMS , domyślna wartość właściwości ASYNC_EXCEPTIONS JMS ConnectionFactory jest zmieniana na ASYNC_EXCEPTIONS_CONNECTIONBROKEN dla IBM MQ classes for JMS. W rezultacie, domyślnie, do aplikacji JMS ExceptionListenerdostarczane są tylko wyjątki odpowiadające kodom błędów zerwanych połączeń.

APAR IT14820, dołączony do produktu IBM MQ 9.0.0 Fix Pack 1, aktualizuje produkt IBM MQ classes for JMS w taki sposób, aby:

- Obiekt ExceptionListener zarejestrowany przez aplikację jest wywoływany dla wszystkich wyjątków zerwanych połączeń, niezależnie od tego, czy aplikacja korzysta z synchronicznych, czy asynchronicznych konsumentów komunikatów.
- Obiekt ExceptionListener zarejestrowany przez aplikację jest wywoływany w przypadku, gdy gniazdo TCP/IP używane przez sesję JMS jest zerwane.
- Wyjątki zerwanych wyjątków (na przykład MQRC_GET_INHIBITED), które pojawiają się podczas dostarczania komunikatów, są dostarczane do aplikacji ExceptionListener aplikacji, gdy aplikacja korzysta z asynchronicznych konsumentów komunikatów, a właściwość ConnectionFactory produktu JMS używana przez aplikację ma właściwość ASYNC_EXCEPTIONS ustawioną na wartość ASYNC_EXCEPTIONS_ALL.

Uwaga: Obiekt ExceptionListener jest wywoływany tylko raz w przypadku wyjątku zerowanego połączenia, nawet jeśli dwa połączenia TCP/IP (jeden używany przez połączenie JMS i jeden używany przez sesję JMS) są zerwane.

W przypadku każdego innego typu problemu wyjątek JMSException jest zgłaszany przez bieżące wywołanie funkcji API produktu JMS .

Jeśli aplikacja nie zarejestrują obiektu nasłuchiwania wyjątków z obiektem połączenia, wszystkie wyjątki, które zostałyby dostarczone do obiektu nasłuchiwania wyjątków, są zapisywane w dzienniku produktu IBM MQ classes for JMS for JMS .

Odsyłacze pokrewne

Klasy produktu IBM MQ dla usługi JMS

WYJĄTEK ASYNCEXCEPTION

Uzyskiwanie dostępu do opcji produktu IBM MQ z poziomu aplikacji IBM MQ classes for JMS

Produkt IBM MQ classes for JMS udostępnia narzędzia do korzystania z wielu funkcji produktu IBM MQ.

Ostrzeżenie: Te funkcje są poza specyfikacją JMS lub, w niektórych przypadkach, naruszają specyfikację JMS . W przypadku ich użycia jest mało prawdopodobne, aby aplikacja była

kompatybilna z innymi dostawcami JMS . Te funkcje, które nie są zgodne ze specyfikacją JMS , są oznaczone za pomocą powiadomienia alarmowego.

Odczytywanie i zapisywanie deskryptora komunikatu z aplikacji IBM MQ classes for JMS
Użytkownik może sterować możliwością uzyskiwania dostępu do deskryptora komunikatu (MQMD), ustawiając właściwości w miejscu docelowym i komunikacie.

Niektóre aplikacje produktu IBM MQ wymagają, aby określone wartości zostały ustawione w strukturze MQMD komunikatów wysyłanych do nich. Produkt IBM MQ classes for JMS udostępnia atrybuty komunikatów, które umożliwiają aplikacjom produktu JMS ustawianie pól MQMD, a więc włączanie aplikacji produktu JMS do "napędu" aplikacji IBM MQ .

Wartość właściwości obiektu docelowego WMQ_MQMD_WRITE_ENABLED należy ustawić na wartość true, aby ustawienie właściwości MQMD miało jakikolwiek wpływ. Następnie można użyć metod ustawiania właściwości komunikatu (na przykład setStringProperty) w celu przypisania wartości do pól MQMD. Wszystkie pola MQMD są prezentowane z wyjątkiem pól StrucId i Version; BackoutCount można odczytać, ale nie zapisywać do.

Ten przykład powoduje, że komunikat jest umieszczany w kolejce lub w temacie MQMD.UserIdentifier jest ustawiony na wartość "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
// ...
```

Przed ustawieniem JMS_IBM_MQMD_UserIdentifiernależy ustawić wartość WMQ_MQMD_MESSAGE_CONTEXT. Więcej informacji na temat korzystania z właściwości WMQ_MQMD_MESSAGE_CONTEXT zawiera sekcja “Właściwości obiektu komunikatu produktu JMS” na stronie 234.

Podobnie można wyodrębnić zawartość pól MQMD, ustawiając wartość WMQ_MQMD_READ_ENABLED na wartość true przed odebraniem komunikatu, a następnie za pomocą metod pobierania komunikatu, na przykład właściwości getString. Wszystkie otrzymane właściwości są przeznaczone tylko do odczytu.

Ten przykład powoduje, że w polu *wartość* znajduje się wartość MQMD.ApplIdentityData : komunikat został zwrócony 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 produktu JMS
Dwie właściwości obiektu docelowego kontrolują dostęp do deskryptora MQMD z produktu JMS, a trzeci kontekst komunikatu steruje kontekstem.

Tabela 35. Nazwy i opisy właściwości

Właściwość	**Postać krótka**	**Opis**
WMQ_MQMD_WRITE_ENABLED	MDW	Określa, czy aplikacja produktu JMS może ustawiać wartości pól MQMD.
WMQ_MQMD_READ_ENABLED	MDR	Określa, czy aplikacja produktu JMS może wyodrębnić wartości pól MQMD.
WMQ_MQMD_MESSAGE_ CONTEXT	MDCTX	Jaki poziom kontekstu komunikatu ma zostać ustawiony przez aplikację JMS . Aby ta właściwość była uwzględniana, aplikacja musi być uruchomiona z odpowiednim uprawnieniem kontekstu.

Tabela 36. Nazwy właściwości, wartości i metody ustawiania

Właściwość	**Poprawne wartości w narzędziu administracyjnym (wartości domyślne są pogrubione)**	**Poprawne wartości w programach**	**Ustaw metodę**
WMQ_MQMD_WRITE (_ENABLED)	• **Nie** Wszystkie właściwości JMS_IBM_MQMD* są ignorowane, a ich wartości nie są kopiowane do bazowej struktury MQMD. • YES Przetwarzane są właściwości JMS_IBM_MQMD*. Ich wartości są kopiowane do bazowej struktury MQMD.	• **Fałsz** • Prawda	setMQMDWriteWłączone

Tabela 36. Nazwy właściwości, wartości i metody ustawiania (kontynuacja)

Właściwość	Poprawne wartości w narzędziu administracyjnym (wartości domyślne są pogrubione)	Poprawne wartości w programach	Ustaw metodę
WMQ_MQMD_READ__ENABLED	• **Nie** Podczas wysyłania komunikatów właściwości JMS_IBM_MQMD* wysłanego komunikatu nie są aktualizowane w celu odzwierciedlenia zaktualizowanych wartości pól w strukturze MQMD. Podczas odbierania komunikatów żadna właściwość JMS_IBM_MQMD* nie jest dostępna w odebranym komunikacie, nawet jeśli nadawca ustawił niektóre lub wszystkie z nich. • YES Podczas wysyłania komunikatów wszystkie właściwości JMS_IBM_MQMD* wysłanego komunikatu są aktualizowane w taki sposób, aby odzwierciedlały zaktualizowane wartości pól w strukturze MQMD, w tym te, które nie zostały jawnie ustawione przez nadawcę. 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** • Prawda	setMQMDReadWłączone

Tabela 36. Nazwy właściwości, wartości i metody ustawiania (kontynuacja)			
Właściwość	**Poprawne wartości w narzędziu administracyjnym (wartości domyślne są pogrubione)**	**Poprawne wartości w programach**	**Ustaw metodę**
WMQ_MQMD _MESSAGE_CONTEXT	• **Domyślnie** Wywołanie funkcji API MQOPEN i struktura MQPMO nie określają jawnych opcji kontekstu komunikatu. • SET_IDENTITY_CONTEXT, Wywołanie funkcji API MQOPEN określa opcję kontekstu komunikatu MQOO_SET_IDENTITY_CONTEXT, a struktura MQPMO określa wartość MQPMO_SET_IDENTITY_CONTEXT. • SET_ALL_CONTEXT Wywołanie funkcji API MQOPEN określa opcję kontekstu komunikatu MQOO_SET_ALL_CONTEXT, a struktura MQPMO określa MQPMO_SET_ALL_CONTEXT.	• **WMQ_MDCTX_DEFAULT** • WMQ_MDCTX_SET_IDENTITY_CONTEXT • WMQ_MDCTX_SET_ALL_CONTEXT	Kontekst setMQMDMessage

Właściwości obiektu komunikatu produktu JMS
Właściwości obiektu komunikatu z przedrostkiem JMS_IBM_MQMD pozwalają na ustawienie lub odczytanie odpowiedniego pola MQMD.

Wysyłanie komunikatów

Wszystkie pola MQMD z wyjątkiem StrucId i Version są reprezentowane. Te właściwości odnoszą się tylko do pól MQMD. W przypadku, gdy właściwość występuje zarówno w strukturze MQMD, jak i w nagłówku MQRFH2 , wersja w tabeli MQRFH2 nie jest ustawiona lub wyodrębniana.

Możliwe jest ustawienie dowolnej z tych właściwości, z wyjątkiem właściwości JMS_IBM_MQMD_BackoutCount. Dowolna wartość ustawiona dla JMS_IBM_MQMD_BackoutCount jest ignorowana.

Jeśli właściwość ma maksymalną długość, a użytkownik poda wartość, która jest zbyt długa, wartość jest obcinana.

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 wartość właściwości WMQ_MQMD_MESSAGE_CONTEXT nie zostanie ustawiona na odpowiednią wartość, wartość właściwości zostanie zignorowana. Jeśli wartość WMQ_MQMD_MESSAGE_CONTEXT zostanie ustawiona na odpowiednią wartość, ale użytkownik nie ma wystarczających uprawnień kontekstowych dla menedżera kolejek, zostanie wygenerowany wyjątek JMSException. Właściwości wymagające konkretnych wartości właściwości WMQ_MQMD_MESSAGE_CONTEXT są następujące.

Następujące właściwości wymagają, aby wartość WMQ_MQMD_MESSAGE_CONTEXT została ustawiona 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 wartość WMQ_MQMD_MESSAGE_CONTEXT została ustawiona na wartość WMQ_MDCTX_SET_ALL_CONTEXT:

- Typ JMS_IBM_MQMD_PutAppl
- Nazwa JMS_IBM_MQMD_PutAppl
- 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 jest ustawiona na wartość true (prawda), bez względu na rzeczywiste właściwości, które zostały ustawione przez aplikację produkujową. Aplikacja nie może modyfikować właściwości odebranego komunikatu, chyba że wszystkie właściwości zostaną usunięte po raz pierwszy zgodnie ze specyfikacją JMS . Odebrany komunikat może zostać przesłany bez modyfikowania właściwości.

Ostrzeżenie: Jeśli aplikacja odbierze komunikat z miejsca docelowego z właściwością WMQ_MQMD_READ_ENABLED ustawioną na wartość true, a następnie przekazuje ją do miejsca docelowego z ustawioną wartością WMQ_MQMD_WRITE_ENABLED na wartość true, spowoduje to, że wszystkie wartości pól MQMD odebranego komunikatu są kopiowane do przekazanego komunikatu.

Tabela właściwości

W tej tabeli znajduje się lista właściwości obiektu komunikatu reprezentującego pola MQMD. Odsyłacze do pełnych opisów pól i ich dozwolonych wartości można znaleźć w sekcji odsyłaczy.

Tabela 37. Nazwy właściwości, opisy i typy

Właściwość	Opis	Java Typ	Odsyłacz do pełnego opisu
Raport JMS_IBM_MQMD_Report	Opcje dla komunikatów raportu	Integer	Raport
JMS_IBM_MQMD_MsgType	Typ komunikatu	Integer	MsgType
JMS_IBM_MQMD_Expiry	Czas życia komunikatu	Integer	Utrata ważności
Sprzężenie zwrotne JMS_IBM_MQMD_Feedback	Informacja zwrotna lub kod przyczyny	Integer	Opinie
Kodowanie JMS_IBM_MQMD_Encoding	Kodowanie numeryczne danych komunikatu	Integer	Kodowanie
JMS_IBM_MQMD_CodedCharSetId	Identyfikator zestawu znaków danych komunikatu	Integer	CodedCharSetId
Format JMS_IBM_MQMD_Format	Nazwa formatu danych komunikatu	Łańcuch	Formatowanie
JMS_IBM_MQMD_Priority [1]	Priorytet komunikatu	Integer	Priorytet
Trwałość JMS_IBM_MQMD_Persistence	Trwałość komunikatu	Integer	Trwałość
JMS_IBM_MQMD_MsgId [2]	Identyfikator komunikatu	Obiekt (byte []) [4]	MsgId
JMS_IBM_MQMD_CorrelId [3]	Identyfikator korelacji	Obiekt (byte []) [4]	CorrelId

Tabela 37. Nazwy właściwości, opisy i typy (kontynuacja)

Właściwość	Opis	Java Typ	Odsyłacz do pełnego opisu
JMS_IBM_MQMD_BackoutCount	Liczba wycofanych	Integer	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
Typ JMS_IBM_MQMD_PutAppl	Typ aplikacji, która wstawiła komunikat	Integer	Typ_aplikacji_wstawiającej
Nazwa JMS_IBM_MQMD_PutAppl	Nazwa aplikacji umieszczonej w komunikacie.	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 dotyczące wniosku dotyczące 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	Integer	Numer_kolejny_komunikatu
Przesunięcie JMS_IBM_MQMD_Offset	Przesunięcie danych w komunikacie fizycznym od początku komunikatu logicznego	Integer	Depozycja
JMS_IBM_MQMD_MsgFlags	Flagi komunikatu	Integer	MsgFlags
JMS_IBM_MQMD_OriginalLength	Długość oryginalnego komunikatu	Integer	OriginalLength

1. **Ostrzeżenie:** Jeśli wartość zostanie przypisana do wartości JMS_IBM_MQMD_Priority, która nie mieści się w zakresie od 0 do 9, to narusza to specyfikację JMS .
2. **Ostrzeżenie:** Specyfikacja JMS wskazuje, że identyfikator komunikatu musi być ustawiony przez dostawcę JMS i musi być unikalny lub ma wartość NULL. Jeśli zostanie przypisana wartość JMS_IBM_MQMD_MsgId, ta wartość zostanie skopiowana do wartości JMSMessageID. Dlatego nie jest on ustawiany przez dostawcę JMS i może nie być unikalny: narusza to specyfikację JMS .

3. **Ostrzeżenie:** Jeśli zostanie przypisana wartość JMS_IBM_MQMD_CorrelId , która rozpoczyna się od łańcucha 'ID:', narusza to specyfikację JMS .

4. **Ostrzeżenie:** Użycie właściwości tablicy bajtów w komunikacie narusza specyfikację JMS .

Uzyskiwanie dostępu do danych komunikatu produktu IBM MQ z aplikacji przy użyciu produktu IBM MQ classes for JMS
Dostęp do pełnych danych komunikatów programu IBM MQ można uzyskać w aplikacji za pomocą programu IBM MQ classes for JMS. Aby uzyskać dostęp do wszystkich danych, komunikat musi być JMSBytesMessage. Treść 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 odebrać wszystkie dane treści komunikatu w produkcie JMSBytesMessage.

Jeśli parametr WMQ_MESSAGE_BODY jest ustawiony na wartość WMQ_MESSAGE_BODY_JMS lub WMQ_MESSAGE_BODY_UNSPECIFIED, treść komunikatu jest zwracana bez nagłówka JMS MQRFH2 , a właściwości składnika JMSBytesMessage odzwierciedlają właściwości ustawione w RFH2.

Niektóre aplikacje nie mogą korzystać z funkcji opisanych w tym temacie. Jeśli aplikacja jest połączona z menedżerem kolejek w wersji IBM MQ V6 lub jeśli dla aplikacji została ustawiona wartość PROVIDERVERSION na 6, funkcje te nie są dostępne.

Wysyłanie wiadomości

Podczas wysyłania komunikatów właściwość miejsca docelowego (WMQ_MESSAGE_BODY) ma pierwszeństwo przed wartością WMQ_TARGET_CLIENT.

Jeśli parametr WMQ_MESSAGE_BODY jest ustawiony na wartość WMQ_MESSAGE_BODY_JMS, produkt IBM MQ classes for JMS automatycznie generuje nagłówek MQRFH2 na podstawie ustawień właściwości i pól nagłówka produktu JMSMessage .

Jeśli parametr WMQ_MESSAGE_BODY jest ustawiony na wartość WMQ_MESSAGE_BODY_MQ, do treści komunikatu nie jest dodawany żaden dodatkowy nagłówek.

Jeśli parametr WMQ_MESSAGE_BODY jest ustawiony na wartość WMQ_MESSAGE_BODY_UNSPECIFIED, program IBM MQ classes for JMS wysyła nagłówek MQRFH2 , chyba że wartość WMQ_TARGET_CLIENT jest ustawiona na wartość WMQ_TARGET_DEST_MQ. Po odebraniu, ustawienie WMQ_TARGET_CLIENT na wartość WMQ_TARGET_DEST_MQ spowoduje usunięcie z treści komunikatu żadnych MQRFH2 .

Uwaga: Produkty JMSBytesMessage i JMSTextMessage nie wymagają MQRFH2, natomiast JMSStreamMessage, JMSMapMessagei JMSObjectMessage .

WMQ_MESSAGE_BODY_UNSPECIFIED jest ustawieniem domyślnym dla WMQ_MESSAGE_BODY, a WMQ_TARGET_DEST_JMS jest domyślnym ustawieniem WMQ_TARGET_CLIENT.

Po wysłaniu JMSBytesMessagemożna przesłonić ustawienia domyślne dla treści komunikatu JMS po utworzeniu komunikatu IBM MQ . Użyj następujących właściwości:

- JMS_IBM_Format lub JMS_IBM_MQMD_Format: ta właściwość określa format nagłówka lub ładunku aplikacji produktu IBM MQ , który uruchamia treść komunikatu produktu JMS , jeśli nie istnieje poprzedni nagłówek WebSphere MQ .
- JMS_IBM_Character_Set lub JMS_IBM_MQMD_CodedCharSetId: ta właściwość określa CCSID ładunku nagłówka lub aplikacji IBM MQ , który uruchamia treść komunikatu JMS , jeśli nie ma poprzedniego nagłówka produktu WebSphere MQ .
- JMS_IBM_Encoding lub JMS_IBM_MQMD_Encoding: ta właściwość określa kodowanie nagłówka lub ładunku aplikacji produktu IBM MQ , które uruchamia treść komunikatu produktu JMS , jeśli nie istnieje poprzedni nagłówek produktu WebSphere MQ .

Jeśli określono oba typy właściwości, właściwości JMS_IBM_MQMD_* przesłaniają odpowiednie właściwości produktu JMS_IBM_* , o ile właściwość docelowa WMQ_MQMD_WRITE_ENABLED jest ustawiona na wartość true.

Różnice w skutkach między ustawieniem właściwości komunikatu za pomocą JMS_IBM_MQMD_* i JMS_IBM_* są znaczące:

1. Właściwości JMS_IBM_MQMD_* są specyficzne dla dostawcy IBM MQ JMS .
2. Właściwości JMS_IBM_MQMD_* są ustawiane tylko w MQMD. Właściwości JMS_IBM_* są ustawiane w MQMD tylko wtedy, gdy komunikat nie ma 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 odbierający prawdopodobnie przyjmie wartości parametrów MQMD.Encoding i MQMD.CodedCharSetId odpowiadają kodowaniu i zestawem znaków liczb i tekstu w treści komunikatu. Jeśli używane są właściwości produktu JMS_IBM_MQMD_* , za pomocą aplikacji wysyłającej należy ją wykonać. Kodowanie i zestaw znaków liczb i tekstu w treści komunikatu są ustawiane za pomocą właściwości JMS_IBM_* .

   Źle zakodowany fragment kodu w produkcie Rysunek 45 na stronie 238 wysyła komunikat zakodowany w zestawie znaków 1208 z zestawem MQMD.CodedCharSetId ustawionym na wartość 37.

   a. Wyślij niesłusznie zakodowaną wiadomość

   ```
   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. Odbierając komunikat, opierając się na wartości JMS_IBM_CHARACTER_SET ustawionej na podstawie wartości MQMD.CodedCharSetId:

   ```
   TextMessage tmi = (TextMessage) cons.receive();
   System.out.println("Message is \"" + tmi.getText() + "\"");
   ```

   c. Wynikowe dane wyjściowe:

   ```
   Message is "éÈÊ'>...??>?"
   ```

   Rysunek 45. Niekonsekwentnie zakodowane dane MQMD i komunikaty

Jeden z fragmentów kodu w produkcie Rysunek 46 na stronie 239 powoduje umieszczenie komunikatu w kolejce lub temacie z jego treścią zawierającą ładunek aplikacji bez dodawania automatycznie wygenerowanego nagłówka MQRFH2 .

1. Ustawianie WMQ_MESSAGE_BODY_MQ:

```
((MQDestination) destination).setMessageBodyStyle
                (WMQConstants.WMQ_MESSAGE_BODY_MQ);
```

2. Ustawianie WMQ_TARGET_DEST_MQ:

```
((MQDestination) destination).setMessageBodyStyle
                (WMQConstants.WMQ_MESSAGE_BODY_UNSPECIFIED);
((MQDestination) destination).
                setTargetClient(WMQConstants.WMQ_TARGET_DEST_MQ);
```

Rysunek 46. Wyślij komunikat z treścią komunikatu MQ .

Odbieranie komunikatu

Jeśli parametr WMQ_MESSAGE_BODY jest ustawiony na wartość WMQ_MESSAGE_BODY_JMS, typ i treść przychodzącego komunikatu JMS są określane przez treść odebranego komunikatu WebSphere MQ . Typ i treść komunikatu są określane przez pola w nagłówku MQRFH2 lub w MQMD(jeśli nie ma MQRFH2).

Jeśli parametr WMQ_MESSAGE_BODY jest ustawiony na wartość WMQ_MESSAGE_BODY_MQ, typ komunikatu przychodzącego JMS to JMSBytesMessage. Treść komunikatu JMS to dane komunikatu zwracane przez bazowe wywołanie funkcji API produktu MQGET . Długość treści komunikatu jest długością zwróconej przez wywołanie MQGET . Zestaw znaków i kodowanie danych w treści komunikatu jest określane za pomocą pól CodedCharSetId i Encoding w MQMD. Format danych w treści komunikatu jest określany na podstawie pola Format w MQMD .

Jeśli właściwość WMQ_MESSAGE_BODY jest ustawiona na wartość WMQ_MESSAGE_BODY_UNSPECIFIED, wartość domyślna IBM MQ classes for JMS ustawia ją na wartość WMQ_MESSAGE_BODY_JMS.

Po odebraniu JMSBytesMessagemożna go zdekodować, odwołując się do następujących właściwości:

- JMS_IBM_Format lub JMS_IBM_MQMD_Format: ta właściwość określa format nagłówka lub ładunku aplikacji produktu IBM MQ , który uruchamia treść komunikatu produktu JMS , jeśli nie istnieje poprzedni nagłówek WebSphere MQ .
- JMS_IBM_Character_Set lub JMS_IBM_MQMD_CodedCharSetId: ta właściwość określa CCSID ładunku nagłówka lub aplikacji IBM MQ , który uruchamia treść komunikatu JMS , jeśli nie ma poprzedniego nagłówka produktu WebSphere MQ .
- JMS_IBM_Encoding lub JMS_IBM_MQMD_Encoding: ta właściwość określa kodowanie nagłówka lub ładunku aplikacji produktu IBM MQ , które uruchamia treść komunikatu produktu JMS , jeśli nie istnieje poprzedni nagłówek produktu WebSphere MQ .

Poniższy fragment kodu powoduje odebraną wiadomość, która jest JMSBytesMessage. Niezależnie od treści odebranego komunikatu i pola formatu odebranego MQMD, komunikat jest JMSBytesMessage.

```
((MQDestination)destination).setMessageBodyStyle
                            (WMQConstants.WMQ_MESSAGE_BODY_MQ);
```

Właściwość miejsca docelowego WMQ_MESSAGE_BODY
Element WMQ_MESSAGE_BODY określa, czy aplikacja JMS przetwarza MQRFH2 komunikatu IBM MQ jako część ładunku komunikatu (czyli jako część treści komunikatu produktu JMS).

Tabela 38. 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 (to znaczy jako część treści komunikatu produktu JMS).

Tabela 39. Nazwy właściwości, wartości i metody ustawiania

Właściwość	Poprawne wartości w narzędziu administracyjnym (wartości domyślne są pogrubione)	Poprawne wartości w programach	Ustaw metodę
WMQ_MESSAGE_BODY	• **Nieokreślone** Podczas wysyłania produkt IBM MQ classes for JMS nie generuje lub nie zawiera nagłówka MQRFH2 , w zależności od wartości właściwości WMQ_TARGET_CLIENT. Podczas odbierania działa jako wartość JMS. • JMS Po wysłaniu produkt IBM MQ classes for JMS automatycznie generuje nagłówek MQRFH2 i dołącza go do komunikatu IBM MQ . Po odebraniu produkt IBM MQ classes for JMS ustawia właściwości komunikatu produktu JMS zgodnie z wartościami w tabeli MQRFH2 (jeśli jest obecny). Nie jest ona obecna w treści komunikatu MQRFH2 jako część treści komunikatu produktu JMS . • MQ Podczas wysyłania program IBM MQ classes for JMS nie generuje MQRFH2. Po odebraniu produkt IBM MQ classes for JMS przedstawia element MQRFH2 jako część treści komunikatu produktu JMS .	• **WMQ_MESSAGE_ BODY_UNSPECIFIED** • WMQ_MESSAGE_BODY_JMS • WMQ_MESSAGE_BODY_MQ	setMessageBodyStyle

Komunikaty trwałe produktu JMS
Aplikacje produktu IBM MQ classes for JMS mogą używać atrybutu kolejki produktu **NonPersistentMessageClass** w celu zapewnienia lepszych wyników dla trwałych komunikatów produktu JMS , kosztem niezawodności.

Kolejka IBM MQ ma atrybut o nazwie **NonPersistentMessageClass**. Wartość tego atrybutu określa, czy nietrwałe komunikaty w kolejce są usuwane po restarcie menedżera kolejek.

Atrybut dla kolejki lokalnej można ustawić za pomocą komendy IBM MQ Script (MQSC), DEFINE QLOCAL, z jednym z następujących parametrów:

NPMCLASS (NORMALNY)
Nietrwałe komunikaty w kolejce są usuwane po restarcie menedżera kolejek. Jest to wartość domyślna.

NPMCLASS (HIGH)
Nietrwałe komunikaty w kolejce nie są usuwane po zrestartowaniu menedżera kolejek po wygaszonym lub natychmiastowym zamknięciu. Komunikaty nietrwałe mogą jednak zostać usunięte po zawłaczającym zamknięciu systemu lub niepowodzeniu.

W tym temacie opisano, w jaki sposób aplikacje produktu IBM MQ classes for JMS mogą używać tego atrybutu kolejki w celu zapewnienia lepszych wyników dla trwałych komunikatów produktu JMS .

Właściwość PERSISTENCE dla obiektu kolejki lub tematu może mieć wartość HIGH. Aby ustawić tę wartość, można użyć narzędzia administracyjnego IBM MQ JMS , a aplikacja może wywołać metodę Destination.setPersistence(), przekazując wartość WMQConstants.WMQ_PER_NPHIGH jako parametr.

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), to komunikat jest umieszczany w kolejce jako komunikat nietrwały IBM MQ . Jeśli właściwość PERSISTENCE dla 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 nietrwałe komunikat JMS jest umieszczany w kolejce jako komunikat nietrwały IBM MQ .

Jeśli trwały komunikat JMS jest umieszczany w kolejce jako nietrwałe komunikat IBM MQ i ma być pewność, że komunikat nie zostanie odrzucony po wygaszonym lub natychmiastowym zamknięciu menedżera kolejek, wszystkie kolejki, przez które komunikat może być kierowany, muszą być ustawione na NPMCLASS (HIGH). W domenie publikowania/subskrybowania kolejki te obejmują kolejki subskrybenta. W ramach pomocy w celu wymuszenia tej konfiguracji produkt 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 dla miejsca docelowego na wartość HIGH nie ma wpływu na sposób odbierania komunikatu z tego miejsca docelowego. Komunikat wysyłany jako trwały komunikat produktu JMS jest odbierany jako trwały komunikat JMS , a komunikat wysyłany jako nietrwały komunikat JMS jest odbierany jako komunikat nietrwały 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 wartość NPMCLASS (HIGH) jest ustawiona w bazowej kolejce produktu IBM MQ . W związku z tym aplikacja musi mieć uprawnienia do wykonywania zapytań w kolejce. Dodatkowo program IBM MQ classes for JMS zachowuje wynik wywołania MQINQ do momentu usunięcia miejsca docelowego i nie wystaw więcej wywołań MQINQ. Dlatego w przypadku zmiany ustawienia NPMCLASS w kolejce bazowej, gdy aplikacja nadal korzysta z miejsca docelowego, program IBM MQ classes for JMS nie zauważe nowego ustawienia.

Zezwalając na umieszczanie trwałych komunikatów produktu JMS w kolejkach produktu IBM MQ jako nietrwałych komunikatów produktu IBM MQ , użytkownik zyskuje wydajność kosztem pewnej niezawodności. Jeśli w przypadku trwałych komunikatów produktu JMS wymagana jest maksymalna

niezawodność, nie należy wysyłać komunikatów do miejsca docelowego, w którym właściwość PERSISTENCE ma wartość HIGH.

Warstwa JMS może używać systemu SYSTEM.JMS.TEMPQ.MODEL, zamiast SYSTEM.DEFAULT.MODEL.QUEUE. SYSTEM.JMS.TEMPQ.MODEL tworzy trwałe kolejki dynamiczne, które akceptują komunikaty trwałe, ponieważ SYSTEM.DEFAULT.MODEL.QUEUE nie można zaakceptować trwałych komunikatów. Aby używać kolejek tymczasowych do akceptowania trwałych komunikatów, należy użyć systemu SYSTEM.JMS.TEMPQ.MODELlub zmień kolejkę modelową na wybraną przez siebie kolejkę alternatywną.

Używanie protokołu TLS z produktem IBM MQ classes for JMS
Aplikacje produktu IBM MQ classes for JMS mogą używać szyfrowania TLS (Transport Layer Security). W tym celu wymagane jest, aby dostawca JSSE był wymagany.

Połączenia produktu IBM MQ classes for JMS przy użyciu protokołu TRANSPORT (CLIENT) obsługują szyfrowanie TLS. Protokół TLS zapewnia szyfrowanie komunikacji, uwierzytelnianie i integralność komunikatów. Jest on zwykle używany do zabezpieczania komunikacji między dowolnymi dwoma równorzędnymi płatnikami w sieci Internet lub w intranecie.

Produkt IBM MQ classes for JMS używa produktu Java Secure Socket Extension (JSSE) do obsługi szyfrowania TLS, a więc wymaga dostawcy JSSE. Maszyny JVM JSE v1.4 mają wbudowaną dostawcę JSSE. Szczegółowe informacje na temat zarządzania certyfikatami i ich przechowywania mogą być różne od dostawcy. Aby uzyskać informacje na ten temat, zapoznaj się z dokumentacją dostawcy JSSE.

W tej sekcji założono, że dostawca JSSE jest poprawnie zainstalowany i skonfigurowany oraz że odpowiednie certyfikaty zostały zainstalowane i udostępnione dla dostawcy JSSE. Teraz można użyć obiektu JMSAdmin, aby ustawić liczbę właściwości administracyjnych.

Jeśli aplikacja IBM MQ classes for JMS używa tabeli definicji kanału klienta (CCDT) do łączenia się z menedżerem kolejek, należy zapoznać się z “Korzystanie z tabeli definicji kanału klienta przy użyciu produktu IBM MQ classes for JMS” na stronie 280.

Właściwość obiektu SSLCIPHERSUITE
Ustaw SSLCIPHERSUITE, aby włączyć szyfrowanie TLS w obiekcie ConnectionFactory .

Aby włączyć szyfrowanie TLS w obiekcie ConnectionFactory , należy użyć obiektu JMSAdmin, aby ustawić właściwość SSLCIPHERSUITE na pakiet CipherSuite obsługiwany przez dostawcę JSSE. Musi to być zgodne ze specyfikacją CipherSpec ustawioną na kanale docelowym. Jednak pakiety CipherSuites różnią się od specyfikacji CipherSpecs i dlatego mają różne nazwy. Produkt “TLS CipherSpecs i CipherSuites w podręczniku IBM MQ classes for JMS” na stronie 245 zawiera tabelę odwzorowania specyfikacji CipherSpecs obsługiwaną przez produkt IBM MQ na równoważną wartość parametru CipherSuites , tak jak jest to znane w przypadku rozszerzenia JSSE. Więcej informacji na temat specyfikacji CipherSpecs i CipherSuites z produktem IBM MQzawiera sekcja Zabezpieczanie produktu IBM MQ.

Na przykład, aby skonfigurować obiekt ConnectionFactory , którego można użyć do utworzenia połączenia przez kanał MQI z włączoną obsługą protokołu TLS z atrybutem CipherSpec o wartości TLS_RSA_WITH_AES_128_CBC_SHA, należy wprowadzić następującą komendę do obiektu JMSAdmin:

```
ALTER CF(my.cf) SSLCIPHERSUITE(SSL_RSA_WITH_AES_128_CBC_SHA)
```

Tę opcję można również ustawić za pomocą aplikacji, używając metody setSSLCipherSuite () w obiekcie MQConnectionFactory .

Dla wygody, jeśli właściwość CipherSpec jest określona we właściwości SSLCIPHERSUITE, JMSAdmin próbuje odwzorować wartość CipherSpec na odpowiedni zestaw CipherSuite i generuje ostrzeżenie. Ta próba odwzorowania nie jest podejmowana, jeśli właściwość jest określona przez aplikację.

Aby zmienić wartość, należy użyć tabeli definicji kanału klienta (CCDT). Więcej informacji na ten temat zawiera sekcja “Korzystanie z tabeli definicji kanału klienta przy użyciu produktu IBM MQ classes for JMS” na stronie 280.

Właściwość obiektu SSLFIPSREQUIRED
Jeśli wymagane jest połączenie z użyciem pakietu CipherSuite obsługiwanego przez dostawcę FIPS IBM Java JSSE (IBMJSSEFIPS), ustaw właściwość SSLFIPSREQUIRED fabryki połączeń na YES.

Wartością domyślną tej właściwości jest NO, co oznacza, że połączenie może używać dowolnego zestawu CipherSuite , który jest obsługiwany przez produkt IBM MQ.

Jeśli aplikacja używa więcej niż jednego połączenia, wartość atrybutu SSLFIPSREQUIRED używana, gdy aplikacja tworzy pierwsze połączenie, określa wartość używaną podczas tworzenia kolejnego połączenia przez aplikację. Oznacza to, że wartość właściwości SSLFIPSREQUIRED fabryki połączeń używanej do tworzenia kolejnego połączenia jest ignorowana. Aby użyć innej wartości parametru SSLFIPSREQUIRED, należy zrestartować aplikację.

Aplikacja może ustawić tę właściwość, wywołując metodę setSSLFipsRequired () obiektu ConnectionFactory . Właściwość ta jest ignorowana, jeśli nie jest ustawiona opcja CipherSuite .

Zadania pokrewne

Określanie, że w czasie wykonywania w kliencie MQI są używane tylko specyfikacje CipherSpecs z certyfikatem FIPS

Odsyłacze pokrewne

Standardy FIPS (Federal Information Processing Standards) dla produktu UNIX, Linux, and Windows

Właściwość obiektu SSLPEERNAME
Użyj funkcji SSLPEERNAME, aby określić wzorzec nazwy wyróżniającej, aby upewnić się, że aplikacja JMS łączy się z poprawnym menedżerem kolejek.

Aplikacja JMS może się upewnić, że łączy się z poprawnym menedżerem kolejek, określając wzorzec nazwy wyróżniającej (DN). Połączenie powiedzie się tylko wtedy, gdy menedżer kolejek przedstawia nazwę wyróżniającą zgodną z wzorcem. Więcej informacji na temat formatu tego wzorca można znaleźć w tematach pokrewnych.

Nazwa wyróżniająca (DN) jest ustawiana przy użyciu właściwości SSLPEERNAME obiektu ConnectionFactory . Na przykład następująca komenda JMSAdmin ustawia obiekt ConnectionFactory , aby oczekiwać, że menedżer kolejek identyfikuje się ze wspólną nazwą rozpoczynającą się od znaków `QMGR.` i z co najmniej dwiema nazwami jednostek organizacyjnych, z których pierwsza musi być IBM i druga WEBSPHERE:

```
ALTER CF(my.cf) SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)
```

Podczas sprawdzania nie jest rozróżniana wielkość liter, a w miejsce przecinków mogą być używane średniki. Parametr SSLPEERNAME można również ustawić z aplikacji przy użyciu metody setSSLPeerName () w obiekcie MQConnectionFactory . Jeśli ta właściwość nie jest ustawiona, sprawdzanie nie jest wykonywane na podstawie nazwy wyróżniającej podanej przez menedżer kolejek. Ta właściwość jest ignorowana, jeśli nie jest ustawiona opcja CipherSuite .

Właściwość obiektu SSLCERTSTORES
Użyj komendy SSLCERTSTORES, aby określić listę serwerów LDAP, które mają być używane na potrzeby sprawdzania listy odwołań certyfikatów (CRL).

Często używana jest lista odwołań certyfikatów (CRL) do identyfikowania certyfikatów, które nie są już zaufane. Listy CRL są zwykle udostępniane na serwerach LDAP. Produkt JMS umożliwia określenie serwera LDAP na potrzeby sprawdzania CRL w systemie Java 2 v1.4 lub nowszym. Następujący przykład JMSAdmin kieruje JMS do korzystania z listy CRL udostępnianej na serwerze LDAP o nazwie crl1.ibm.com:

```
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com)
```

Uwaga: Aby pomyślnie użyć narzędzia CertStore z listą CRL udostępnianą na serwerze LDAP, należy upewnić się, że pakiet Java Software Development Kit (SDK) jest kompatybilny z 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 zamiast niego RFC 2256.

Jeśli serwer LDAP nie jest uruchomiony na domyślnym porcie 389, można określić port, dodając dwukropek (:) i numer portu do nazwy hosta. Jeśli certyfikat prezentowany przez menedżera kolejek znajduje się na liście CRL udostępnionej na stronie crl1.ibm.com, połączenie nie zostanie zakończone. Aby uniknąć pojedynczego punktu awarii, produkt JMS zezwala na dostarczanie wielu serwerów LDAP, dostarczając listę serwerów LDAP z ogranicznikami o charakterze 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 na kolei, dopóki nie znajdzie serwera, z którym będzie mógł pomyślnie zweryfikować certyfikat menedżera kolejek. Każdy serwer musi zawierać identyczne informacje.

Łańcuch w tym formacie może być dostarczony przez aplikację w metodzie MQConnectionFactory.setSSLCertStores (). Alternatywnie aplikacja może utworzyć jedną 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 list CRL. Szczegółowe informacje na temat konstruowania i używania obiektów CertStore można znaleźć w dokumentacji JSSE.

Poprawność certyfikatu prezentowanego przez menedżera kolejek przy ustawionym połączeniu jest sprawdzana w następujący sposób:

1. Pierwszy obiekt CertStore w kolekcji identyfikowanej przez sslCertStores służy do identyfikacji serwera CRL.
2. Podjęto próbę skontaktowania się z serwerem CRL.
3. Jeśli próba zakończy się pomyślnie, serwer jest wyszukiwany pod kątem zgodności z certyfikatem.
   a. Jeśli certyfikat zostanie unieważniony, proces wyszukiwania zakończy się, a żądanie nawiązania połączenia nie powiedzie się i zostanie odebrany kod przyczyny MQRC_SSL_CERTIFICATE_ODWOŁANE.
   b. Jeśli certyfikat nie zostanie znaleziony, proces wyszukiwania zostanie nadany, a połączenie może być kontynuowane.
4. Jeśli próba skontaktowania się z serwerem nie powiedzie się, następny obiekt CertStore jest używany do identyfikacji serwera CRL, a proces jest powtarzany z kroku 2.

   Jeśli był to ostatni element CertStore w kolekcji lub jeśli kolekcja nie zawiera obiektów CertStore , proces wyszukiwania nie powiódł się i żądanie połączenia nie powiedzie się i kod przyczyny MQRC_SSL_CERT_STORE_ERROR.

Obiekt Collection określa kolejność, w jakiej używane są CertStores .

Jeśli aplikacja korzysta z funkcji setSSLCertStores () w celu ustawienia kolekcji obiektów CertStore , obiekt MQConnectionFactory nie może być już powiązany z przestrzenią nazw JNDI. Próba wykonania tego zadania powoduje wystąpienie wyjątku. Jeśli właściwość sslCertStores nie jest ustawiona, sprawdzanie odwołań nie jest wykonywane na certyfikacie udostępnionym przez menedżer kolejek. Ta właściwość jest ignorowana, jeśli nie jest ustawiona opcja CipherSuite .

Właściwość obiektu SSLRESETCOUNT
Ta właściwość reprezentuje łączną liczbę bajtów wysłanych i odebranych przez połączenie, zanim klucz tajny używany do szyfrowania jest ponownie negocjowany.

Liczba wysłanych bajtów jest liczbą przed zaszyfrowaniem, a liczba odebranych bajtów jest liczbą po deszyfrowaniu. Liczba bajtów obejmuje również informacje sterujące wysłane i odebrane przez program IBM MQ classes for JMS.

Na przykład, aby skonfigurować obiekt ConnectionFactory , który może być używany do tworzenia połączenia przez kanał MQI z włączoną obsługą protokołu TLS z kluczem tajnym, który jest ponownie negocjowany po 4 MB danych, należy wprowadzić następującą komendę do obiektu JMSAdmin:

```
ALTER CF(my.cf) SSLRESETCOUNT(4194304)
```

Aplikacja może ustawić tę właściwość, wywołując metodę setSSLResetCount () obiektu ConnectionFactory .

Jeśli wartość tej właściwości jest równa zero, która jest wartością domyślną, klucz tajny nigdy nie jest ponownie negocjowany. Właściwość ta jest ignorowana, jeśli nie jest ustawiona opcja CipherSuite .

Właściwość obiektu SSLSocketFactory
Aby dostosować inne aspekty połączenia TLS dla aplikacji, należy utworzyć fabrykę SSLSocketFactory i skonfigurować produkt JMS w taki sposób, aby go używała.

Użytkownik może dostosować inne aspekty 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 jest odpowiednio dostosowany. Informacje na temat tego, jak to zrobić, można znaleźć w dokumentacji JSSE, ponieważ konfigurowalne składniki różnią się od dostawcy. Po uzyskaniu odpowiedniego obiektu SSLSocketFactory należy użyć metody MQConnectionFactory.setSSLSocketFactory (), aby skonfigurować produkt JMS w taki sposób, aby używany był dostosowany obiekt SSLSocketFactory .

Jeśli aplikacja korzysta z metody setSSLSocketFactory () w celu ustawienia dostosowanego obiektu SSLSocketFactory , obiekt MQConnectionFactory nie może być już powiązany z przestrzenią nazw JNDI. Próba wykonania tego zadania powoduje wystąpienie wyjątku. Jeśli ta właściwość nie jest ustawiona, zostanie użyty domyślny obiekt SSLSocketFactory . Szczegółowe informacje na temat zachowania domyślnego obiektu SSLSocketFactory można znaleźć w dokumentacji rozszerzenia JSSE. Ta właściwość jest ignorowana, jeśli nie jest ustawiona opcja CipherSuite .

Ważne: Nie należy zakładać, że użycie właściwości SSL zapewnia bezpieczeństwo, gdy obiekt ConnectionFactory jest pobierany z przestrzeni nazw JNDI, która nie jest sama w sobie zabezpieczona. W szczególności standardowa implementacja protokołu LDAP w interfejsie JNDI nie jest zabezpieczona. Atakujący może naśladować serwer LDAP, wprowadzając w błąd aplikację JMS do łączenia się z niewłaściwym serwerem bez zauważania. Dzięki odpowiednim uzgodnieniom zabezpieczeń inne implementacje interfejsu JNDI (takie jak implementacja protokołu fscontext) są zabezpieczone.

Wprowadzanie zmian w magazynie kluczy JSSE lub magazynie zaufanych certyfikatów
W przypadku wprowadzenia zmian w magazynie kluczy lub magazynie zaufanych certyfikatów należy podjąć określone działania, aby zmiany zostały pobrane.

Jeśli zawartość magazynu kluczy lub magazynu zaufanych certyfikatów JSSE zostanie zmieniona lub zostanie zmieniona lokalizacja pliku kluczy lub pliku zaufanych certyfikatów, wówczas aplikacje produktu IBM MQ classes for JMS , które są uruchomione w danym momencie, nie pobierają automatycznie zmian. Aby zmiany zostały uwzględnione, muszą zostać wykonane następujące działania:

- Aplikacje muszą zamknąć wszystkie połączenia i zniszczyć wszelkie nieużywane połączenia w pulach połączeń.
- Jeśli dostawca JSSE buforuje informacje z magazynu kluczy i magazynu zaufanych certyfikatów, te informacje muszą zostać odświeżone.

Po wykonaniu tych czynności aplikacje mogą następnie ponownie utworzyć połączenia.

W zależności od sposobu zaprojektowania aplikacji oraz funkcji udostępnianej przez dostawcę JSSE możliwe jest wykonanie tych działań bez zatrzymywania i restartowania aplikacji. Jednak zatrzymywanie i restartowanie aplikacji może być najprostszym rozwiązaniem.

TLS CipherSpecs i CipherSuites w podręczniku IBM MQ classes for JMS
Zdolność aplikacji IBM MQ classes for JMS do nawiązywania połączeń z menedżerem kolejek zależy od specyfikacji CipherSpec określonej na końcu serwera kanału MQI i CipherSuite określonej na końcu klienta.

W poniższej tabeli znajduje się lista specyfikacji CipherSpecs obsługiwanych przez produkt IBM MQ oraz ich odpowiedniki CipherSuites.

Należy przejrzeć temat Nieaktualne CipherSpecs , aby sprawdzić, czy którykolwiek z CipherSpecs, wymieniony w poniższej tabeli, jest nieaktualny w produkcie IBM MQ , a jeśli tak, to przy czym aktualizacja specyfikacji CipherSpec była nieaktualna.

Ważne: wymienione w liście CipherSuites są obsługiwane przez środowisko wykonawcze IBM Java Runtime Environment (JRE) dostarczane z produktem IBM MQ. Wymienione pakiety CipherSuites obejmują te, które są obsługiwane przez środowisko JRE firmy Oracle Java . Więcej informacji na temat konfigurowania aplikacji do korzystania ze środowiska Oracle Java JRE zawiera sekcja Konfigurowanie aplikacji w celu używania odwzorowań IBM Java lub Oracle Java CipherSuite.

Tabela wskazuje również protokół używany do komunikacji, niezależnie od tego, czy pakiet CipherSuite jest zgodny ze standardem FIPS 140-2.

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 zgodność ze standardem FIPS 140-2 została skonfigurowana dla aplikacji (patrz następujące uwagi w konfiguracji), można skonfigurować tylko te CipherSuites , które są oznaczone jako zgodne ze standardem FIPS 140-2. Próba użycia innych pakietów CipherSuites powoduje błąd.

Uwaga: Każde środowisko JRE może mieć wiele dostawców zabezpieczeń kryptograficznych, z których każdy może przyczyniać się do implementacji tego samego pakietu CipherSuite. Jednak nie wszyscy dostawcy zabezpieczeń są certyfikowani zgodnie ze standardem FIPS 140-2. Jeśli zgodność ze standardem FIPS 140-2 nie jest wymuszana dla aplikacji, możliwe jest użycie niecertyfikowanej implementacji pakietu CipherSuite . Niecertyfikowane implementacje mogą nie działać zgodnie ze standardem FIPS 140-2, nawet jeśli teoretycznie CipherSuite spełnia minimalny poziom zabezpieczeń wymagany przez standard. Więcej informacji na temat konfigurowania wymuszania FIPS 140-2 w aplikacjach IBM MQ JMS zawierają następujące uwagi.

Więcej informacji na temat zgodności ze standardem FIPS 140-2 i Suite-B dla specyfikacji CipherSpecs i CipherSuiteszawiera sekcja Określanie specyfikacji CipherSpecs. Konieczne może być również zapoznanie się z informacjami, które dotyczą Stanów Zjednoczonych Federal Information Processing Standards(Federal Information Processing Standards).

Aby korzystać z pełnego zestawu pakietów CipherSuites i obsługiwać zgodność ze standardem FIPS 140-2 i/lub Suite-B, wymagane jest odpowiednie środowisko JRE. IBM Produkt Java 7 Service Refresh 4 z pakietem poprawek Fix Pack 2 lub wyższy poziom środowiska IBM JRE zapewnia odpowiednią obsługę protokołu TLS 1.2 CipherSuites wymienionego w sekcji Tabela 40 na stronie 247.

Aby możliwe było użycie szyfrowania TLS v1.3 , środowisko JRE, na którym działa aplikacja, musi obsługiwać protokół TLS v1.3.

Uwaga: Aby można było używać niektórych pakietów CipherSuites, w środowisku JRE należy skonfigurować pliki strategii 'nieograniczone'. Więcej szczegółowych informacji na temat konfigurowania plików strategii w pakiecie SDK lub środowisku JRE zawiera temat *Pliki strategii pakietu SDK produktu IBM* w publikacji *Skorowidz zabezpieczeń dla pakietu IBM SDK, Java Technology Edition* dla używanej wersji.

Tabela 40. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**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 40. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**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 40. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	Odpowiednik CipherSuite (IBM JRE)	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 40. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	Odpowiednik CipherSuite (IBM JRE)	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 40. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	Odpowiednik CipherSuite (IBM JRE)	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 40. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**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

Tabela 40. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**Odpowiednik CipherSuite (Oracle JRE)**	**Protokół**	**Zgodny ze standardem FIPS 140-2**
ECDHE_ECDSA_RC4_128_SHA256	SSL_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS 1.2	no

Tabela 40. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**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 40. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	Odpowiednik CipherSuite (IBM JRE)	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 40. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**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 40. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**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 40. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**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 40. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	Odpowiednik CipherSuite (IBM JRE)	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

Tabela 40. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**Odpowiednik CipherSuite (Oracle JRE)**	**Protokół**	**Zgodny ze standardem FIPS 140-2**
ECDHE_RSA_RC4_128_SHA256	SSL_ECDHE_RSA_WITH_RC4_128_SHA	TLS_ECDHE_RSA_WITH_RC4_128_SHA	TLS 1.2	no

Tabela 40. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	Odpowiednik CipherSuite (IBM JRE)	Odpowiednik CipherSuite (Oracle JRE)	Protokół	Zgodny ze standardem FIPS 140-2
TLS_RSA_WITH_3DES_EDE_CBC_SHA "1" na stronie 274	SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0	yes

Tabela 40. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	Odpowiednik CipherSuite (IBM JRE)	Odpowiednik CipherSuite (Oracle JRE)	Protokół	Zgodny ze standardem FIPS 140-2
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	yes

Tabela 40. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**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	yes

Tabela 40. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**Odpowiednik CipherSuite (Oracle JRE)**	**Protokół**	**Zgodny ze standardem FIPS 140-2**
TLS_RSA_WITH_AES_128_GCM_SHA256	SSL_RSA_WITH_AES_128_GCM_SHA256	TLS_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	yes

Tabela 40. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**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	yes

Tabela 40. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	Odpowiednik CipherSuite (IBM JRE)	Odpowiednik CipherSuite (Oracle JRE)	Protokół	Zgodny ze standardem FIPS 140-2
TLS_RSA_WITH_AES_256_CBC_SHA256	SSL_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2	yes

Tabela 40. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**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	yes

Tabela 40. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**Odpowiednik CipherSuite (Oracle JRE)**	**Protokół**	**Zgodny ze standardem FIPS 140-2**
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	TLS 1.0	no

Tabela 40. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	Odpowiednik CipherSuite (IBM JRE)	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

Tabela 40. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**Odpowiednik CipherSuite (Oracle JRE)**	**Protokół**	**Zgodny ze standardem FIPS 140-2**
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 40. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	Odpowiednik CipherSuite (IBM JRE)	Odpowiednik CipherSuite (Oracle JRE)	Protokół	Zgodny ze standardem FIPS 140-2
V 9.1.5 TLS_AES_128_GCM_SHA256 “2” na stronie 275	TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256	TLS 1.3	no
V 9.1.5 TLS_AES_256_GCM_SHA384 “2” na stronie 275	TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384	TLS 1.3	no

Tabela 40. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**Odpowiednik CipherSuite (Oracle JRE)**	**Protokół**	**Zgodny ze standardem FIPS 140-2**
V 9.1.5 TLS_CHACHA20_POLY1305_SHA256 "2" na stronie 275	TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256	TLS 1.3	no

Tabela 40. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	Odpowiednik CipherSuite (IBM JRE)	Odpowiednik CipherSuite (Oracle JRE)	Protokół	Zgodny ze standardem FIPS 140-2
V 9.1.5 TLS_AES_128_CCM_SHA256 "2" na stronie 275	TLS_AES_128_CCM_SHA256	TLS_AES_128_CCM_SHA256	TLS 1.3	no
V 9.1.5 TLS_AES_128_CCM_8_SHA256 "2" na stronie 275	TLS_AES_128_CCM_8_SHA256	TLS_AES_128_CCM_8_SHA256	TLS 1.3	no

Tabela 40. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	Odpowiednik CipherSuite (IBM JRE)	Odpowiednik CipherSuite (Oracle JRE)	Protokół	Zgodny ze standardem FIPS 140-2
V 9.1.5 Dowolna "2" na stronie 275	*ANY	*ANY	Wielokrotny	no
V 9.1.5 ANY_TLS13 "2" na stronie 275	*TLS13	*TLS13	TLS V13	no
V 9.1.5 ANY_TLS12_OR_HIGHER "2" na stronie 275	*TLS12ORHIGHER	*TLS12ORHIGHER	TLS V1.2 i nowsze	no
V 9.1.5 ANY_TLS13_OR_HIGHER "2" na stronie 275	*TLS13ORHIGHER	*TLS13ORHIGHER	TLS V1.3 i nowsze	no

Uwagi:

1. Klasa CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA jest nieaktualna. Jednak może być ona nadal używana do przesyłania do 32 GB danych, zanim połączenie zostanie zakończone z błędem AMQ9288.

Aby uniknąć tego błędu, należy unikać używania potrójnego algorytmu szyfrowania DES lub włączyć resetowanie klucza tajnego podczas korzystania z tej specyfikacji CipherSpec.

2. V 9.1.5 Aby możliwe było użycie szyfrowania TLS v1.3 , środowisko JRE, na którym działa aplikacja, musi obsługiwać protokół TLS v1.3 .

Konfigurowanie pakietów Ciphersuites i zgodności ze standardem FIPS w aplikacji IBM MQ classes for JMS

- Aplikacja, która używa produktu IBM MQ classes for JMS , może użyć jednej z dwóch metod w celu ustawienia pakietu CipherSuite dla połączenia:
  - Wywołaj metodę pakietu setSSLCipherSuite obiektu ConnectionFactory .
  - Za pomocą narzędzia administracyjnego IBM MQ JMS ustaw właściwość SSLCIPHERSUITE obiektu ConnectionFactory .
- Aplikacja, która używa produktu 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 .
  - Za pomocą narzędzia administracyjnego IBM MQ JMS można ustawić właściwość SSLFIPSREQUIRED obiektu ConnectionFactory .

Konfigurowanie aplikacji do korzystania z odwzorowań IBM Java lub Oracle Java CipherSuite

Istnieje możliwość skonfigurowania, czy aplikacja używa domyślnych odwzorowań IBM Java CipherSuite do IBM MQ CipherSpec , czy Oracle CipherSuite do IBM MQ CipherSpec odwzorowań. Oznacza to, że można używać protokołu TLS CipherSuites , niezależnie od tego, czy aplikacja korzysta ze środowiska IBM JRE, czy środowiska Oracle JRE. Właściwość systemowa Java `com.ibm.mq.cfg.useIBMCipherMappings` określa, które odwzorowania są używane. Właściwość może mieć jedną z następujących wartości:

true
Użyj opcji IBM Java CipherSuite do IBM MQ CipherSpec odwzorowań.

Ta wartość jest wartością domyślną.

Fałsz
Use the Oracle CipherSuite to IBM MQ CipherSpec mappings.

Więcej informacji na temat korzystania z programów IBM MQ Java i TLS Ciphers zawiera blog MQdev: MQ Java, TLS Ciphers, Non-IBM JREs & APARs IT06775, IV66840, IT09423, IT10837.

Ograniczenia współdziałania

Niektóre pakiety CipherSuites mogą być kompatybilne z więcej niż jednym IBM MQ CipherSpec, w zależności od tego, jaki protokół jest używany. Jednak obsługiwane jest tylko połączenie CipherSuite/CipherSpec , które używa wersji TLS określonej w tabeli 1. Próba użycia nieobsługiwanych kombinacji CipherSuites i CipherSpecs nie powiedzie się i zostanie zgłoszony odpowiedni wyjątek. Instalacje korzystające z dowolnego z tych kombinacji CipherSuite/CipherSpec powinny przenieść się do obsługiwanej kombinacji.

W poniższej tabeli przedstawiono pakiety CipherSuites , do których ma zastosowanie to ograniczenie.

Tabela 41. CipherSuites i ich obsługiwane i nieobsługiwane CipherSpecs		
CipherSuite	**Obsługiwany protokół TLS CipherSpec**	**Nieobsługiwany protokół SSL CipherSpec**
SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA "1" na stronie 276	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. Ta specyfikacja CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA jest nieaktualna. Jednak może być ona nadal używana do przesyłania do 32 GB danych, zanim połączenie zostanie zakończone z błędem AMQ9288. Aby uniknąć tego błędu, należy unikać używania potrójnego algorytmu szyfrowania DES lub włączyć resetowanie klucza tajnego podczas korzystania z tej specyfikacji CipherSpec.

Zapisywanie wyjść kanału w produkcie Java dla produktu IBM MQ classes for JMS
Wyjścia kanału można utworzyć, definiując klasy produktu Java implementujące określone interfejsy.

W pakiecie com.ibm.mq.exits są zdefiniowane trzy interfejsy:

- WMQSendExit, dla wyjścia wysyłania
- WMQReceiveExit, dla wyjścia odbierania
- WMQSecurityExit, dla wyjścia zabezpieczeń

Poniższy przykładowy kod 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 otrzymuje jako parametry obiekt MQCXP i obiekt MQCD. Obiekty te reprezentują struktury MQCXP i MQCD zdefiniowane w interfejsie proceduralnym.

Gdy wywoływane jest wyjście wysyłania, parametr agentBuffer zawiera dane, które mają zostać wysłane do menedżera kolejek serwera. Parametr długości nie jest wymagany, ponieważ wyrażenie agentBuffer.limit () udostępnia 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 wysyłania wyjść, zwrócone dane są przekazywane do następnego wyjścia wysyłania w sekwencji. Wyjście wysyłania może zwrócić zmodyfikowaną wersję danych, które otrzymuje w parametrze agentBuffer , lub może zwrócić dane bez zmian. Najprostszym możliwym organem wyjściowym jest zatem:

```
{ return agentBuffer; }
```

Po wywołaniu wyjścia odbierania parametr agentBuffer zawiera dane, które zostały odebrane z menedżera kolejek serwera. Wyjście odbierania zwraca jako wartość dane, które mają być przekazane do aplikacji przez program IBM MQ classes for JMS. Jeśli jednak wyjście odbierania nie jest ostatnim wyjściem odbioru w sekwencji wyjść odbierania, zwrócone dane są przekazywane do następnego wyjścia odbierania w sekwencji.

Po wywołaniu wyjścia zabezpieczeń parametr agentBuffer zawiera dane, które zostały odebrane w przepływie zabezpieczeń od wyjścia zabezpieczeń na końcu serwera połączenia. Wyjście zabezpieczeń zwraca jako wartość dane, które mają zostać wysłane w przepływie zabezpieczeń do wyjścia zabezpieczeń serwera.

Wyjścia kanału są wywoływane z buforem, w którym znajduje się tablica zapasowa. Aby uzyskać najlepszą wydajność, wyjście powinno zwrócić bufor z tablicą bazową.

Do wyjścia kanału można przekazać do 32 znaków danych użytkownika, gdy jest on wywoływany. Wyjście uzyskuje dostęp do danych użytkownika, wywołując metodę getExitData () obiektu MQCXP. Chociaż wyjście może zmienić dane użytkownika, wywołując metodę setExitData (), dane użytkownika są odświeżane za każdym razem, gdy wywoływane jest wyjście. W związku z tym wszelkie zmiany wprowadzone w danych użytkownika są tracone. Wyjście może jednak przekazywać dane z jednego wywołania do następnego przy użyciu obszaru użytkownika wyjścia obiektu MQCXP. Wyjście uzyskuje dostęp do obszaru użytkownika wyjścia przez odwołanie, wywołując metodę getExitUserArea().

Każda klasa wyjścia musi mieć konstruktor. Konstruktorem może być konstruktor domyślny, tak jak pokazano to w poprzednim przykładzie, lub konstruktor z parametrem łańcuchowym. Konstruktor jest wywoływany w celu utworzenia instancji klasy wyjścia dla każdego wyjścia zdefiniowanego w klasie. Dlatego w poprzednim przykładzie instancja klasy MyMQExits jest tworzona dla wyjścia wysyłania, inna instancja jest tworzona dla wyjścia odbierania, a dla wyjścia zabezpieczeń tworzona jest trzecia instancja. Gdy wywoływany jest konstruktor z parametrem łańcuchowym, parametr ten 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 zamykać połączenia z poziomu wyjścia kanału.

Po wysłaniu danych do końca połączenia z serwerem, szyfrowanie TLS jest wykonywane *po* wywołaniu wszystkich wyjść kanału. Podobnie, gdy dane są odbierane z końca połączenia serwera, deszyfrowanie TLS jest wykonywane *przed* wywołaniem dowolnego wyjścia kanału.

W wersjach produktu IBM MQ classes for JMS wcześniejszych niż 7.0wyjścia kanału zostały zaimplementowane 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 kanału
Aplikacja IBM MQ classes for JMS może używać zabezpieczeń kanału, wysyłania i odbierania wyjść z kanału MQI, który jest uruchamiany, gdy aplikacja łączy się z menedżerem kolejek. Aplikacja może używać wyjść napisanych w Java, C lub C + +. Aplikacja może również używać sekwencji wyjść wysyłania lub odbierania, które są uruchamiane w ramach dziedziczenia.

Następujące właściwości są używane przez określenie wyjścia wysyłania lub sekwencji wyjść wysyłania używanych przez połączenie JMS :

- Właściwość **SENDEXIT** obiektu MQConnectionFactory .
- Właściwość **sendexit** w specyfikacji aktywowania używanej przez adapter zasobów produktu IBM MQ na potrzeby komunikacji przychodzącej,

- Właściwość **sendexit** w obiekcie ConnectionFactory używanym przez adapter zasobów IBM MQ do komunikacji wyjściowej.

Wartość właściwości jest łańcuchem składowym, który składa się z co najmniej jednego elementu rozdzielonego przecinkami. Każdy element identyfikuje wyjście wysyłania w jeden z następujących sposobów:

- Nazwa klasy, która implementuje interfejs `WMQSendExit` dla wyjścia wysyłania napisanego w produkcie Java.
- Łańcuch w formacie *libraryName* (*entryPointnazwa*) dla wyjścia wysyłania napisanego w języku C lub C + +.

W podobny sposób, następujące właściwości określają wyjście odbierania lub sekwencję wyjść odbierania, używane przez połączenie:

- Właściwość **RECEXIT** obiektu MQConnectionFactory .
- Właściwość **receiveexit** w specyfikacji aktywowania używanej przez adapter zasobów produktu IBM MQ na potrzeby komunikacji przychodzącej,
- Właściwość **receiveexit** w obiekcie ConnectionFactory używanym 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 produktu IBM MQ na potrzeby komunikacji przychodzącej,
- Właściwość **securityexit** w obiekcie ConnectionFactory używanym przez adapter zasobów IBM MQ do komunikacji wyjściowej.

W przypadku właściwości MQConnectionFactoriesmożna ustawić właściwości **SENDEXIT**, **RECEXIT** i **SECEXIT** , używając narzędzia administracyjnego IBM MQ JMS lub 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 lokalizacje w podanej kolejności.

1. Ścieżka klasy określona przez właściwość **com.ibm.mq.cfg.ClientExitPath.JavaExitsClasspath** lub przez atrybut **JavaExitsClassPath** w sekcji Kanały w pliku konfiguracyjnym klienta IBM MQ .
2. Ścieżka klasy określona przez właściwość systemową Java **com.ibm.mq.exitClasspath**. Należy pamiętać, że ta właściwość jest już nieaktualna.
3. Katalog programu IBM MQ kończy działanie, jak pokazano na rysunku Tabela 42 na stronie 278. 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 następnie pliki JAR w katalogu.

Tabela 42. Katalog wyjść programu IBM MQ

Platforma	Katalog
Linux UNIX UNIX and Linux	`/var/mqm/exits` (32-bitowe wyjścia kanału) `/var/mqm/exits64` (64-bitowe wyjścia kanału)
Windows Windows	*katalog_danych_instalacji*\exits gdzie *katalog_danych_instalacji* to katalog, który został wybrany dla plików danych programu IBM MQ podczas instalacji. Katalog domyślny to C:\ProgramData\IBM\MQ.

Uwaga: Jeśli wyjście kanału istnieje w więcej niż jednej lokalizacji, program IBM MQ classes for JMS ładuje pierwszą instancję, która zostanie znaleziona.

Elementem nadrzędnym programu ładującego klasy jest program ładujący klasy, który jest używany do ładowania IBM MQ classes for JMS. Dlatego program ładujący klasy macierzyste może załadować wyjście kanału, jeśli nie można go znaleźć w żadnej z poprzednich lokalizacji. Jeśli jednak produkt IBM MQ classes for JMS jest używany w środowisku, takim jak serwer aplikacji JEE , prawdopodobnie nie ma możliwości wywierania wpływu na wybór nadrzędnego programu ładującego klasy, dlatego program ładujący klasy powinien zostać skonfigurowany przez ustawienie właściwości systemowej Java **com.ibm.mq.cfg.ClientExitPath.JavaExitsClasspath** na serwerze aplikacji.

Jeśli aplikacja jest uruchamiana z włączoną obsługą Java security manager , plik konfiguracyjny strategii używany przez środowisko wykonawcze produktu Java , w którym działa aplikacja, musi mieć uprawnienia do ładowania klasy wyjścia kanału. Więcej informacji na ten temat zawiera sekcja Uruchamianie klas produktu IBM MQ dla aplikacji JMS w ramach menedżera zabezpieczeń Java.

Interfejsy `MQSendExit`, `MQReceiveExit`i `MQSecurityExit` dostarczane z wersjami produktu IBM WebSphere MQ wcześniejszymi niż IBM WebSphere MQ 7.0 są nadal obsługiwane. Jeśli używane są wyjścia kanału, które implementują te interfejsy, produkt `com.ibm.mq.jar` musi być obecny w ścieżce klasy.

Informacje na temat pisania wyjść kanału w języku C zawiera sekcja “Programy obsługi wyjścia kanału dla kanałów przesyłania komunikatów” na stronie 1059. Należy zapisać programy obsługi wyjścia kanału napisane w języku C lub C + + w katalogu podanym w sekcji Tabela 42 na stronie 278.

Jeśli aplikacja korzysta z tabeli definicji kanału klienta (CCDT) w celu nawiązania połączenia z menedżerem kolejek, należy zapoznać się z “Korzystanie z tabeli definicji kanału klienta przy użyciu produktu IBM MQ classes for JMS” na stronie 280.

Określanie danych użytkownika, które mają być przekazywane do wyjść kanału podczas korzystania z produktu IBM MQ classes for JMS
Do wyjścia kanału można przekazać do 32 znaków danych użytkownika, gdy jest on wywoływany.

Właściwość SENDEXITINIT obiektu MQConnectionFactory określa dane użytkownika, które są przekazywane do każdego wyjścia wysyłania podczas jego wywołania. Wartość właściwości to łańcuch składający się z co najmniej jednego elementu danych użytkownika oddzielonych przecinkami. Pozycja każdego elementu danych użytkownika w łańcuchu określa, które wyjście wysyłania, w sekwencji wysyłania wyjść, dane użytkownika są przekazywane do. Na przykład pierwsza pozycja danych użytkownika w łańcuchu jest przekazywana do pierwszego wyjścia wysyłania w sekwencji wysyłania wyjść.

Właściwość SENDEXITINIT można ustawić za pomocą narzędzia administracyjnego IBM MQ JMS lub IBM MQ Explorer. Alternatywnie, aplikacja może ustawić właściwość, wywołując metodę setSendExitInit().

W podobny sposób właściwość RECEXITINIT obiektu ConnectionFactory określa dane użytkownika, które są 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 produktu IBM MQ Explorer. Alternatywnie, aplikacja może ustawić właściwości, wywołując metody setReceiveExitInit() i setSecurityExitInit().

Podczas określania danych użytkownika, które są przekazywane do wyjść kanału, należy pamiętać o następujących regułach:

- Jeśli liczba elementów danych użytkownika w łańcuchu jest większa niż liczba wyjść w sekwencji, nadmiarowe pozycje 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 przecinki w spadku w łańcuchu lub przecinek na początku łańcucha oznaczają również nieokreślony element danych użytkownika.

Jeśli aplikacja korzysta z tabeli definicji kanału klienta (CCDT) w celu nawiązania połączenia z menedżerem kolejek, wszystkie dane użytkownika określone w definicji kanału połączenia klienta są przekazywane do wyjść kanału, gdy są wywoływane. Więcej informacji na temat korzystania z tabeli

definicji kanału klienta zawiera sekcja “Korzystanie z tabeli definicji kanału klienta przy użyciu produktu IBM MQ classes for JMS” na stronie 280.

Korzystanie z tabeli definicji kanału klienta przy użyciu produktu IBM MQ classes for JMS
Aplikacja IBM MQ classes for JMS może korzystać z definicji kanału połączenia klienta zapisanych w tabeli definicji kanału klienta (CCDT). W celu użycia tabeli definicji kanału klienta należy skonfigurować obiekt ConnectionFactory . Istnieją pewne ograniczenia dotyczące jego stosowania.

Alternatywnie do tworzenia definicji kanału połączenia klienta przez ustawienie określonych właściwości obiektu ConnectionFactory aplikacja IBM MQ classes for JMS może używać definicji kanału połączenia klienta, które są przechowywane w tabeli definicji kanału klienta. Definicje te są tworzone za pomocą komend IBM MQ Script (MQSC) lub IBM MQ Programmable Command Format (PCF). Gdy aplikacja tworzy obiekt połączenia, program IBM MQ classes for JMS przeszukuje tabelę definicji kanału klienta pod kątem odpowiedniej definicji kanału połączenia klienckiego i używa definicji kanału w celu uruchomienia kanału MQI. Więcej informacji na temat tabel definicji kanału klienta i sposobu ich tworzenia zawiera sekcja Tabela definicji kanału klienta.

Aby można było używać tabeli definicji kanału klienta, właściwość CCDTURL obiektu ConnectionFactory musi być ustawiona na obiekt URL. Program IBM MQ classes for JMS nie odczytuje informacji o tabeli definicji kanału klienta z pliku konfiguracyjnego produktu IBM MQ MQI client , ale niektóre inne wartości są z niego używane (patrz temat “Plik konfiguracyjny IBM MQ classes for JMS” na stronie 95 , dla którego ma zastosowanie wartość). Obiekt URL hermetyzuje jednolity wskaźnik zasobów (URL), który identyfikuje nazwę i położenie pliku zawierającego tabelę definicji kanału klienta i określa sposób dostępu do pliku. Właściwość CCDTURL można ustawić za pomocą narzędzia administracyjnego IBM MQ JMS , a aplikacja może ustawić właściwość, tworząc obiekt URL i wywołując metodę setCCDTURL() obiektu ConnectionFactory .

Jeśli na przykład plik ccdt1.tab zawiera tabelę definicji kanału klienta i jest przechowywany w tym samym systemie, w którym działa aplikacja, to aplikacja może ustawić właściwość CCDTURL w następujący sposób:

```
java.net.URL chanTab1 = new URL("file:///home/admdata/ccdt1.tab");
factory.setCCDTURL(chanTab1);
```

Inny przykład: przypuśćmy, 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, których można użyć dla parametru **QMgrName** w wywołaniu MQCONN wywoływanym przez aplikację kliencką korzystaniową z interfejsu kolejki komunikatów (Message Queue Interface-MQI). Więcej informacji na temat znaczenia tych wartości znajduje się w sekcji MQCONN. Właściwość QMANAGER można ustawić za pomocą narzędzia administracyjnego IBM MQ JMS lub programu IBM MQ Explorer. Alternatywnie, aplikacja może ustawić właściwość, wywołując metodę setQueueManager () obiektu ConnectionFactory .

Jeśli aplikacja utworzy 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 w celu wyszukania odpowiedniej definicji kanału połączenia klienckiego, a następnie używa definicji kanału w celu uruchomienia kanału MQI w menedżerze kolejek.

Należy zauważyć, że właściwości CCDTURL i CHANNEL obiektu ConnectionFactory nie mogą być ustawione w obu przypadkach, gdy aplikacja wywołuje metodę createConnection(). Jeśli ustawione

są obie właściwości, metoda zgłasza wyjątek. Właściwość CCDTURL lub CHANNEL jest uznawana za ustawioną, jeśli jej wartością jest dowolna wartość inna niż null, pusty łańcuch lub łańcuch zawierający wszystkie puste znaki.

Gdy program IBM MQ classes for JMS znajdzie odpowiednią definicję kanału połączenia klienta w tabeli definicji kanału klienta, używa ona tylko informacji wyodrębnionych z tabeli w celu uruchomienia kanału MQI. Wszystkie właściwości powiązane z kanałem obiektu ConnectionFactory są ignorowane.

W szczególności, jeśli używany jest protokół TLS, należy zwrócić uwagę na następujące kwestie:

- Kanał MQI używa protokołu TLS tylko wtedy, gdy definicja kanału wyodrębniona z tabeli definicji kanału klienta określa nazwę klasy CipherSpec obsługiwaną przez produkt IBM MQ classes for JMS.
- Tabela definicji kanału klienta zawiera również informacje na temat położenia serwerów LDAP (Lightweight Directory Access Protocol), które przechowują listy odwołań certyfikatów (CRL). Produkt IBM MQ classes for JMS używa tylko tych informacji w celu uzyskania dostępu do serwerów LDAP, które przechowują listy CRL.
- Tabela definicji kanału klienta może również zawierać położenie respondera OCSP. Produkt IBM MQ classes for JMS nie może używać informacji OCSP z pliku tabeli definicji kanału klienta. Można jednak skonfigurować protokół OCSP zgodnie z opisem w sekcji Protokół OCSP (Online Certificate Status Protocol) w aplikacjach klienckich Java i JMS.

Więcej informacji na temat używania protokołu TLS z tabelą definicji kanału klienta zawiera sekcja Korzystanie z rozszerzonego klienta transakcyjnego z kanałami TLS.

Jeśli używane są wyjścia kanału, należy pamiętać również o następujących punktach:

- Kanał MQI korzysta tylko z wyjść kanału i powiązanych danych użytkownika określonych przez definicję kanału wyodrębnioną z tabeli definicji kanału klienta.
- Definicja kanału wyodrębniona z tabeli definicji kanału klienta może określać wyjścia kanału, które są zapisywane w produkcie Java. Oznacza to, na przykład, że parametr SCYEXIT w komendzie DEFINE CHANNEL w celu utworzenia definicji kanału połączenia klienta może określać nazwę klasy implementujący interfejs WMQSecurityExit . Podobnie, parametr SENDEXIT może określać nazwę klasy implementujący interfejs WMQSendExit , a parametr RCVEXIT może określać nazwę klasy implementujący interfejs WMQReceiveExit . Więcej informacji na temat pisania wyjścia kanału w programie Javazawiera sekcja“Zapisywanie wyjść kanału w produkcie Java dla produktu IBM MQ classes for JMS” na stronie 276.

  Obsługiwane jest również użycie wyjść kanału napisanych w języku innym niż Java . Więcej informacji na temat określania parametrów SCYEXIT, SENDEXIT i RCVEXIT w komendzie DEFINE CHANNEL dla wyjść kanału napisanych w innym języku zawiera sekcja DEFINE CHANNEL(DEFINIOWANIE KANAŁU).

Automatyczne ponowne połączenie klienta JMS
Skonfiguruj klienta produktu JMS w taki sposób, aby automatycznie łączył się z nim automatycznie po awarii sieci, menedżera kolejek lub serwera.

Zwykle, jeśli autonomiczna aplikacja produktu IBM MQ classes for JMS jest połączona z menedżerem kolejek przy użyciu transportu klienta, a menedżer kolejek staje się niedostępny z jakiegoś powodu (na przykład z powodu wyłączenia sieci, awarii menedżera kolejek lub zatrzymanego menedżera kolejek), produkt IBM MQ classes for JMS zgłosi wyjątek JMSException podczas następnej próby nawiązania komunikacji przez aplikację z menedżerem kolejek. Aplikacja musi wychwycić wyjątek JMSException i podjąć próbę ponownego nawiązania połączenia z menedżerem kolejek. Użytkownik może uprościć projektowanie aplikacji, 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 połączenia.

Korzystanie z tej implementacji automatycznego ponownego nawiązywania połączenia z klientem nie jest obsługiwane w ramach serwerów aplikacji Java Platform, Enterprise Edition . Alternatywną implementację można znaleźć w sekcji “Korzystanie z automatycznego ponownego połączenia klienta w środowiskach Java EE” na stronie 288 .

Korzystanie z automatycznego ponownego połączenia klienta JMS
Jeśli autonomiczna aplikacja produktu IBM MQ classes for JMS korzysta z fabryki połączeń, która ma ustawioną właściwość CONNECTIONNAMELIST lub CCDTURL, to aplikacja może korzystać z automatycznego ponownego nawiązywania połączenia z klientem.

Automatyczne ponowne połączenie klienta może być używane do ponownego połączenia z menedżerami kolejek, w tym również z tymi, które są częścią konfiguracji wysokiej dostępności (HA). Konfiguracje wysokiej dostępności obejmują menedżery kolejek z wieloma instancjami, menedżery kolejek RDQM lub menedżery kolejek wysokiej dostępności na urządzeniu IBM MQ .

Zachowanie funkcji automatycznego ponownego połączenia klienta, które jest udostępniane przez produkt IBM MQ classes for JMS , zależy od właściwości, które są następujące:

TRANSPORT właściwości fabryki połączeń produktu JMS (krótka nazwa TRAN)

TRANSPORT określa, w jaki sposób aplikacje, które korzystają z fabryki połączeń, łączą się z menedżerem kolejek. Ta właściwość musi być ustawiona na wartość CLIENT dla automatycznego ponownego połączenia klienta, które ma być używane. Automatyczne ponowne połączenie klienta nie jest dostępne dla aplikacji, które łączą się z menedżerem kolejek używający fabryki połączeń, która ma właściwość TRANSPORT ustawioną na BIND, DIRECT lub DIRECTHTTP.

Właściwość fabryki połączeń produktu JMS QMANAGER (krótka nazwa QMGR)

Właściwość QMANAGER określa nazwę menedżera kolejek, z którym łączy się fabryka połączeń.

Właściwość JMS fabryki połączeń 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 podczas korzystania z transportu CLIENT. Lista ma następujący format: nazwa hosta (port), nazwa hosta (port).

Właściwość CCDTURL fabryki połączeń produktu JMS (krótka nazwa CCDT)

Właściwość CCDTURL wskazuje tabelę definicji kanału klienta, która jest używana przez produkt IBM MQ classes for JMS podczas nawiązywania połączenia z menedżerem kolejek przy użyciu tabeli definicji kanału klienta.

Właściwość fabryki połączeń produktu JMS CLIENTRECONNECTOPTIONS (krótka nazwa CROPT)

Parametr CLIENTRECONNECTOPTIONS określa, czy produkt IBM MQ classes for JMS będzie próbował automatycznie połączyć się z menedżerem kolejek w imieniu aplikacji, jeśli menedżer kolejek stanie się dostępny.

Atrybut DefRecon w sekcji Kanały w pliku konfiguracyjnym klienta

Atrybut DefRecon udostępnia opcję administracyjną, która umożliwia automatyczne ponowne łączenie wszystkich aplikacji lub wyłączenie automatycznego ponownego połączenia dla aplikacji, które są zapisywane w celu automatycznego ponownego nawiązywania połączenia.

Automatyczne ponowne połączenie klienta jest dostępne tylko wtedy, gdy aplikacja pomyślnie łączy się z menedżerem kolejek.

Gdy aplikacja łączy się z menedżerem kolejek, który korzysta z transportu CLIENT, IBM MQ classes for JMS używa wartości właściwości fabryki połączeń CLIENTRECONNECTOPTIONS w celu określenia, czy ma być używane automatyczne ponowne połączenie klienta, czy menedżer kolejek, z którym połączona jest aplikacja, staje się niedostępny. W tabeli 1 przedstawiono możliwe wartości dla właściwości CLIENTRECONNECTOPTIONS, a także zachowanie IBM MQ classes for JMS dla każdej z następujących wartości:

Tabela 43. Możliwe wartości właściwości CLIENTRECCECTOPTIONS.

CLIENTRECONNECTOPTIONS	**Zachowanie produktu IBM MQ classes for JMS**
ANY	Jeśli ustawiona jest wartość CONNECTIONNAMELIST, należy użyć wartości właściwości CONNECTIONNAMELIST, aby otworzyć połączenie z nazwą hosta i kombinacją portów, a następnie połączyć się z dowolnym menedżerem kolejek. Aby można było używać tej opcji automatycznego ponownego połączenia z klientem, właściwość QMANAGER musi być ustawiona na wartość domyślną lub "*". Jeśli ustawiono wartość CCDTURL, otwórz tabelę definicji kanału klienta, która jest określona przez właściwość CCDTURL, wybierz pozycję w tabeli, a następnie użyj tej pozycji, aby uruchomić kanał połączenia klienta z menedżerem kolejek. Aby można było używać tej opcji automatycznego ponownego połączenia z klientem, właściwość QMANAGER musi być ustawiona na: • Gwiazdka (*) • Gwiazdka (*), po której następuje nazwa grupy menedżerów kolejek • Pusty łańcuch lub łańcuch zawierający wszystkie puste znaki.
ASDEF	Aby określić, czy automatyczne ponowne połączenie klienta jest dostępne, należy użyć wartości DefRecon .
WYŁĄCZONE	Nie wykonuj żadnego automatycznego ponownego połączenia klienta i zwróć wyjątek JMSException do aplikacji.
QMGR	Określa, że klient musi ponownie nawiązać połączenie z tym samym menedżerem kolejek. Ta opcja musi być używana w przypadku rozwiązań wysokiej dostępności, w przypadku których wymagane jest ponowne połączenie z inną instancją tego samego menedżera kolejek. Jeśli ustawiona jest wartość CONNECTIONNAMELIST, należy użyć wartości właściwości CONNECTIONNAMELIST w celu otwarcia połączenia z nazwą hosta i kombinacją portów, a następnie połączyć się z menedżerem kolejek określonym przez właściwość QMANAGER. Jeśli ustawiono wartość CCDTURL, otwórz tabelę definicji kanału klienta, która jest określona przez właściwość CCDTURL, znajdź pozycje w tabeli, które są zgodne z nazwą menedżera kolejek określoną przez właściwość QMANAGER, a następnie użyj tych pozycji, aby uruchomić kanał połączenia klienta z tym menedżerem kolejek.

Jeśli ustawiona jest wartość CONNECTIONNAMELIST, podczas automatycznego ponownego połączenia klienta produkt IBM MQ classes for JMS używa informacji znajdujących się na liście CONNNECTIONNAMELIST właściwości fabryki połączeń w celu określenia systemu, z którym ma zostać ponownie nawiązane połączenie.

Początkowo program IBM MQ classes for JMS próbuje ponownie nawiązać połączenie, używając nazwy hosta i portu określonego w pierwszej pozycji na liście CONNECTIONNAMELIST. Jeśli nawiąże połączenie, program IBM MQ classes for JMS podejmie próbę nawiązania połączenia z menedżerem kolejek, który ma nazwę określoną we właściwości QMANAGER. Jeśli można nawiązać połączenie z menedżerem kolejek, program IBM MQ classes for JMS ponownie otwiera wszystkie obiekty IBM MQ , które były otwarte przez aplikację przed automatycznym ponownym połączeniem klienta i kontynuują działanie tak, jak wcześniej.

Jeśli połączenie nie może zostać nawiązane z wymaganym menedżerem kolejek przy użyciu pierwszej pozycji na liście CONNECTIONNAMELIST, program IBM MQ classes for JMS próbuje drugą pozycję na liście CONNECTIONNAMELIST, itd.

Gdy program IBM MQ classes for JMS wypróbowuje wszystkie pozycje z listy CONNECTIONNAMELIST, czekają one na pewien okres czasu, zanim ponownie spróbują ponownie nawiązać połączenie. Aby wykonać nową próbę ponownego nawiązania połączenia, IBM MQ classes for JMS rozpoczyna się od pierwszej pozycji na liście CONNECTIONNAMELIST. Następnie wszystkie pozycje są wykonywane na liście CONNECTIONNAMELIST z kolei do czasu ponownego nawiązania połączenia lub osiągnięcia końca listy CONNECTIONNAMELIST, co oznacza, że serwer IBM MQ classes for JMS czeka przez pewien okres czasu, zanim podejmie próbę ponownego nawiązania połączenia.

Jeśli parametr CCDTURL jest ustawiony, podczas automatycznego ponownego nawiązywania połączenia klient IBM MQ classes for JMS używa tabeli definicji kanału klienta określonej we właściwości CCDTURL, aby określić, do jakiego systemu ma zostać ponownie nawiązane połączenie.

IBM MQ classes for JMS początkowo analizuje tabelę definicji kanału klienta i znajduje odpowiedni wpis, który jest zgodny z wartością właściwości QMANAGER. Po znalezieniu pozycji program IBM MQ classes for JMS podejmuje próbę ponownego nawiązania połączenia z wymaganym menedżerem kolejek przy użyciu tego wpisu. Jeśli można nawiązać połączenie z menedżerem kolejek, program IBM MQ classes for JMS ponownie otwiera wszystkie obiekty IBM MQ , które były otwarte przez aplikację przed automatycznym ponownym połączeniem klienta i kontynuują działanie tak, jak wcześniej.

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 użyć tego połączenia itd.

Po wypróbowaniu przez program IBM MQ classes for JMS wszystkich odpowiednich pozycji w tabeli definicji kanału klienta, przed ponowną próbą ponownego nawiązania połączenia czekają one na pewien okres. Aby wykonać nową próbę ponownego nawiązania połączenia, IBM MQ classes for JMS ponownie analizuje tabelę definicji kanału klienta i próbuje uzyskać pierwszą odpowiednią pozycję. Następnie wypróbowują każdą odpowiednią pozycję w tabeli definicji kanału klienta z kolei do czasu ponownego nawiązania połączenia lub próby ostatniego odpowiedniego wpisu w tabeli definicji kanału klienta. W tym momencie program IBM MQ classes for JMS czeka na pewien okres czasu przed ponowną próbą wykonania operacji.

Niezależnie od tego, czy używany jest parametr CONNECTIONNAMELIST, czy CCDTURL, proces automatycznego ponownego nawiązywania połączenia z klientem jest kontynuowany do momentu pomyślnego nawiązania połączenia z menedżerem kolejek IBM MQ classes for JMS przez właściwość QMANAGER do momentu pomyślnego nawiązania połączenia z menedżerem kolejek.

Domyślnie próby ponownego połączenia są podejmowane w następujących odstępach czasu:

- Pierwsza próba jest wykonana po początkowym opóźnieniu wynoszącym 1 sekundę, plus losowy element do 250 milisekund.
- Druga próba to 2 sekundy, plus losowy odstęp do 500 milisekund, po pierwszej próbie awarii.
- Trzecia próba to 4 sekundy, plus losowy odstęp czasu do 1 sekundy, po drugiej próbie awarii.
- Czwarta próba wykonana jest 8 sekund, plus przypadkowy odstęp czasu do 2 sekund, po trzeciej próbie awarii.

- Piąta próba jest wykonana 16 sekund, plus przypadkowy odstęp czasu do 4 sekund, po czwartej próbie nie udaje się.
- Szósta próba, a wszystkie kolejne próby są wykonane 25 sekund, plus przypadkowy odstęp czasu do 6 sekund i 250 milisekund po poprzedniej próbie awarii.

Próby ponownego połączenia są opóźnione w odstępach czasu, które są częściowo stałe i częściowo losowe. Ma to zapobiec wszystkim aplikacjom produktu IBM MQ classes for JMS , które były połączone z menedżerem kolejek, który nie jest już dostępny z ponownego połączenia jednocześnie.

Jeśli konieczne jest zwiększenie wartości domyślnych, aby dokładniej odzwierciedlić czas wymagany do odtworzenia menedżera kolejek lub aktywnego menedżera kolejek, należy zmodyfikować atrybut ReconDelay w sekcji Channel pliku konfiguracyjnego klienta. Więcej informacji na ten temat zawiera sekcja Sekcja CHANNELS w pliku konfiguracyjnym klienta.

To, czy aplikacja IBM MQ classes for JMS nadal działa poprawnie po ponownym połączeniu, zależy od jego projektu. Zapoznaj się z tematami pokrewnymi, aby dowiedzieć się, w jaki sposób projektować aplikacje mogą korzystać z funkcji automatycznego ponownego połączenia.

Kody przyczyny wskazujące, że menedżer kolejek nie jest już dostępny
Kody przyczyn wskazują, że menedżer kolejek nie jest już dostępny lub nie może zostać osiągnięty podczas próby automatycznego ponownego nawiązywania połączenia z produktem IBM MQ classes for JMS .

Produkt “Automatyczne ponowne połączenie klienta JMS” na stronie 281 zawiera przegląd wyjątków JMSExceptions oraz sposób automatycznego restartowania aplikacji, a informacje zawarte w sekcji “Korzystanie z automatycznego ponownego połączenia klienta JMS” na stronie 282 -wymagania dotyczące automatycznego ponownego nawiązywania połączenia z klientem.

Poniższe informacje zawierają listę kodów przyczyny produktu IBM MQ , które powinny zostać sprawdzone przez aplikację:

RC2009
MQRC_CONNECTION_BROKEN

RC2059
MQRC_Q_MGR_NOT_AVAILABLE

RC2161
MQRC_Q_MGR_QUIESCING,

RC2162
MQRC_Q_MGR_ZATRZYMYWANIE

RC2202
MQRC_CONNECTION_QUIESCING

RC2203
MQRC_CONNECTION_ZATRZYMYWANIE

RC2223
MQRC_Q_MGR_NOT_ACTIVE

RC2279
MQRC_CHANNEL_STOPPED_BY_USER

RC2537
MQRC_CHANNEL_NOT_AVAILABLE

RC2538
MQRC_HOST_NOT_AVAILABLE

Większość wyjątków JMSExceptions, które są zgłaszane z powrotem do aplikacji korporacyjnych, zawiera powiązany wyjątek MQException, który zawiera kod przyczyny. Aby zaimplementować logikę ponawiania dla kodów przyczyny znajdujących się na poprzedniej liście, aplikacje korporacyjne powinny sprawdzić ten powiązany wyjątek przy użyciu kodu podobnego do następującego 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
            }
        }
    }
```

Odsyłacze pokrewne

Klasy produktu IBM MQ dla usługi JMS

Korzystanie z automatycznego ponownego połączenia klienta w środowiskach Java SE i Java EE

Istnieje możliwość użycia automatycznego ponownego połączenia klienta IBM MQ w celu ułatwienia różnych rozwiązań wysokiej dostępności (HA) i odtwarzania po awarii (DR) w środowisku Java SE i Java EE .

Różne rozwiązania HA i DR są dostępne na różnych platformach:

- Multi Menedżery kolejek o wielu instancjach są instancjami tego samego menedżera kolejek skonfigurowanego na różnych serwerach (patrz sekcja Menedżery kolejek z wieloma instancjami). Jedna instancja menedżera kolejek jest definiowana jako aktywna instancja, a inna instancja jest zdefiniowana jako instancja rezerwowa. Jeśli aktywna instancja nie powiedzie się, menedżer kolejek z wieloma instancjami zostanie zrestartowany automatycznie na serwerze rezerwowym.

  Zarówno aktywne, jak i rezerwowe menedżery kolejek mają ten sam identyfikator menedżera kolejek (QMID). Aplikacje klienckie produktu IBM MQ , które łączą się z menedżerem kolejek z wieloma instancjami, można skonfigurować w taki sposób, aby automatycznie ponownie nawiązały połączenie z instancją rezerwową menedżera kolejek przy użyciu automatycznego ponownego nawiązywania połączenia z klientem.
- Linux RDQM (replikowany menedżer kolejek danych) jest rozwiązaniem wysokiej dostępności, które jest dostępne na platformach Linux (patrz sekcja Wysoka dostępność RDQM). Konfiguracja RDQM składa się z trzech serwerów skonfigurowanych w grupie wysokiej dostępności (HA), a każda z nich jest instancją menedżera kolejek. Jedną z instancji jest działający menedżer kolejek, który synchronicznie replikuje swoje dane do pozostałych dwóch instancji. Jeśli serwer, na którym działa ten menedżer kolejek, nie powiedzie się, inna instancja menedżera kolejek zostanie uruchomiona i będzie miała bieżące dane, z którymi będzie działać. Trzy instancje menedżera kolejek współużytkują zmienny adres IP, dlatego klienty muszą być skonfigurowane tylko za pomocą pojedynczego adresu IP. Aplikacje klienckie, które łączą się z menedżerem kolejek RDQM, można skonfigurować w taki sposób, aby automatycznie ponownie nawiązały połączenie z instancją rezerwową menedżera kolejek za pomocą automatycznego ponownego nawiązywania połączenia z klientem.
- MQ Appliance Rozwiązanie wysokiej dostępności może być również dostarczane przez parę IBM MQ urządzeń (patrz Wysoka dostępność i Disaster Recovery w dokumentacji produktu IBM MQ Appliance). Menedżer kolejek wysokiej dostępności jest uruchamiany na jednym z urządzeń, podczas synchronicznego replikowania danych do rezerwowej instancji menedżera kolejek na drugim urządzeniu. Jeśli urządzenie podstawowe nie powiedzie się, menedżer kolejek zostanie automatycznie uruchomiony i uruchomiony na innym urządzeniu. Dwie instancje menedżera kolejek można skonfigurować w taki sposób, aby współużytkował zmienny adres IP, dlatego klienty muszą być skonfigurowane tylko przy użyciu pojedynczego adresu IP. Aplikacje klienckie, które łączą się z menedżerem kolejek wysokiej dostępności na urządzeniu IBM MQ , można skonfigurować w taki sposób, aby automatycznie ponownie nawiązały połączenie z instancją rezerwową menedżera kolejek za pomocą automatycznego ponownego nawiązywania połączenia z klientem.

Pojęcia pokrewne

Menedżery kolejek z wieloma instancjami
Automatyczne ponowne łączenie klienta
rdqm wysoka dostępność

Korzystanie z automatycznego ponownego połączenia klienta w środowiskach Java SE
Aplikacje korzystające z programu IBM MQ classes for JMS działającego w środowiskach Java SE mogą korzystać z funkcji automatycznego ponownego połączenia klienta za pośrednictwem właściwości fabryki połączeń **CLIENTRECONNECTOPTIONS**.

Właściwość fabryki połączeń **CLIENTRECONNECTOPTIONS** korzysta z dwóch dodatkowych właściwości fabryki połączeń, **CONNECTIONNAMELIST** i **CCDTURL**, w celu określenia sposobu łączenia się z serwerem, na którym jest uruchomiony menedżer kolejek.

CONNECTIONNAMELIST property

Właściwość **CONNECTIONNAMELIST** jest listą rozdzielaną przecinkami zawierającą informacje o nazwie hosta i porcie, które mają być używane do łączenia się z menedżerem kolejek w trybie klienta. Ta właściwość jest używana z wartościami **QMANAGER** i **CHANNEL** . Gdy aplikacja korzysta z właściwości **CONNECTIONNAMELIST** w celu utworzenia połączenia klienckiego, program IBM MQ classes for JMS próbuje połączyć się z każdym hostem w kolejności wyświetlania listy. Jeśli pierwszy host menedżera kolejek jest niedostępny, program IBM MQ classes for JMS podejmie próbę nawiązania połączenia z następnym hostem na liście. Jeśli koniec listy nazw połączeń zostanie osiągnięty bez utworzenia połączenia, IBM MQ classes for JMS zgłosi kod przyczyny `MQRC_QMGR_NOT_AVAILABLE` IBM MQ .

Jeśli menedżer kolejek, z którym połączona jest aplikacja, nie powiedzie się, wszystkie aplikacje, które korzystały z programu **CONNECTIONNAMELIST** w celu nawiązania połączenia z tym menedżerem kolejek, otrzymują wyjątek wskazujący, że menedżer kolejek jest niedostępny. Aplikacja musi wychwycić wyjątek i wyczyścić wszystkie zasoby, które były używane. Aby utworzyć połączenie, aplikacja musi korzystać z fabryki połączeń. Fabryka połączeń próbuje ponownie nawiązać połączenie z każdym hostem w kolejności listy, menedżer kolejek, który uległ awarii, nie jest teraz dostępny. Fabryka połączeń próbuje połączyć się z innym hostem na liście.

CCDTURL property

Właściwość **CCDTURL** zawiera adres URL (Uniform Resource Locator) wskazujący na tabelę definicji kanału klienta (CCDT). Właściwość ta jest używana z właściwością **QMANAGER** . Pakiet CCDT zawiera listę kanałów klienta, które są używane do łączenia się z menedżerem kolejek zdefiniowanym w systemie IBM MQ . Informacje o tym, w jaki sposób CCDT są używane przez produkt IBM MQ classes for JMS, zawiera sekcja “Korzystanie z tabeli definicji kanału klienta przy użyciu produktu IBM MQ classes for JMS” na stronie 280.

Korzystanie z właściwości CLIENTRECONNECTOPTIONS w celu włączenia automatycznego ponownego połączenia klienta w obrębie partycji IBM MQ classes for JMS

Właściwość **CLIENTRECONNECTOPTIONS** służy do włączania automatycznego ponownego nawiązywania połączenia z klientem w obrębie partycji IBM MQ classes for JMS. Możliwe wartości dla tej właściwości są następujące:

ASDEF
Zachowanie automatycznego ponownego połączenia klienta jest definiowane przez wartość domyślną określoną w sekcji kanału w pliku konfiguracyjnym klienta IBM MQ (`mqclient.ini`).

WYŁĄCZONE
Automatyczne ponowne połączenie klienta jest wyłączone.

QMGR
IBM MQ classes for JMS podejmuje próbę nawiązania połączenia z menedżerem kolejek o tym samym identyfikatorze menedżera kolejek, co menedżer kolejek, z którym był połączony, przy użyciu jednej z następujących opcji:

- Właściwość **CONNECTIONNAMELIST** i kanał, który jest zdefiniowany we właściwości **CHANNEL** .
- Środowisko CCDT zdefiniowane we właściwości **CCDTURL** .

ANY
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 w pliku konfiguracyjnym klienta

Korzystanie z automatycznego ponownego połączenia klienta w środowiskach Java EE
Adapter zasobów produktu IBM MQ , który można wdrożyć w środowiskach Java EE (Java Platform, Enterprise Edition), a dostawca przesyłania komunikatów produktu WebSphere Application Server IBM MQ używa produktu IBM MQ classes for JMS do komunikowania się z menedżerami kolejek produktu IBM MQ . The IBM MQ resource adapter and the WebSphere Application Server IBM MQ messaging provider provide support for automatic client reconnection.

Opcje dostępne w celu zapewnienia automatycznego ponownego połączenia klienta w środowisku Java EE to:

- Specyfikacja aktywowania
- Porty nasłuchiwania produktu WebSphere Application Server
- Korporacyjne komponenty JavaBeans i aplikacje z interfejsem WWW
- Aplikacje działające w kontenerach klienta

Uwaga: Automatyczne ponowne połączenie klienta ze specyfikacjami aktywowania przy użyciu funkcji udostępnianej przez produkt IBM MQ classes for JMS nie jest obsługiwane. Adapter zasobów produktu IBM MQ udostępnia własny mechanizm służący do ponownego łączenia specyfikacji aktywowania, jeśli menedżer kolejek, z którym łączy się specyfikacja aktywowania, staje się niedostępny.

Mechanizm ten jest kontrolowany przez:

- The IBM MQ resource adapter property **reconnectionRetryCount**.
- The IBM MQ resource adapter property **reconnectionRetryInterval**.
- Właściwość specyfikacji aktywowania **connectionNameList**.

Więcej informacji na temat tych właściwości zawiera sekcja “Konfiguracja właściwości obiektu ResourceAdapter” na stronie 456.

Użycie automatycznego ponownego połączenia klienta w ramach metody onMessage() aplikacji komponentu bean sterowanego komunikatami lub innej aplikacji działającej w środowisku produktu Java Platform, Enterprise Edition nie jest obsługiwane. Jeśli menedżer kolejek, z którym nawiąże połączenie, staje się niedostępny, aplikacja musi zaimplementować własną logikę ponownego połączenia.

Obsługa automatycznego ponownego nawiązywania połączenia z klientem w środowiskach Java EE
Within Java EE environments, such as WebSphere Application Server, the IBM MQ resource adapter and the WebSphere Application Server IBM MQ messaging provider provide support for automatic client reconnection. Jednak w niektórych przypadkach do tego wsparcia mają zastosowanie ograniczenia.

Adapter zasobów produktu IBM MQ , który może zostać wdrożony w środowiskach Java EE i dostawcy przesyłania komunikatów produktu WebSphere Application Server IBM MQ , umożliwia komunikację z menedżerami kolejek produktu IBM MQ za pomocą konsoli IBM MQ classes for JMS .

The following table summarizes the support that the IBM MQ resource adapter and the WebSphere Application Server IBM MQ messaging provider provide support for automatic client reconnection.

Tabela 44. Podsumowanie obsługi automatycznych opcji ponownego połączenia klienta w środowiskach Java EE

Opcje automatycznego ponownego połączenia	**Właściwość CONNECTIONNAMELIST**	**CCDTURL, właściwość**	**Właściwość CLIENTRECONNECTOPTIONS**	**Alternatywne podejście do automatycznego ponownego nawiązywania połączenia z klientem**
Specyfikacje aktywowania	Obsługiwane z ograniczeniami	Obsługiwane z ograniczeniami	Nieobsługiwane	Specyfikacje środowiska i aktywowania produktu Java EE udostępniają własny mechanizm ponownego połączenia.
Porty nasłuchiwania produktu WebSphere Application Server	Obsługiwane z ograniczeniami	Obsługiwane z ograniczeniami	Nieobsługiwane	Produkt WebSphere Application Server udostępnia własny mechanizm ponownego połączenia.
Korporacyjne komponenty JavaBeans i aplikacje z interfejsem WWW	Obsługiwane z ograniczeniami	Obsługiwane z ograniczeniami	Nieobsługiwane	Aplikacja musi implementować własną logikę ponownego połączenia
Aplikacje działające w kontenerach klienta	Obsługiwany	Obsługiwany	Obsługiwany	Nie dotyczy

Aplikacje komponentów bean sterowane komunikatami, które są instalowane w środowisku Java EE , takie jak IBM MQ classes for JMS, mogą używać specyfikacji aktywowania do przetwarzania komunikatów w systemie IBM MQ . Specyfikacje aktywowania służą do wykrywania komunikatów docierających do systemu IBM MQ i dostarczania ich do komponentów bean sterowanych komunikatami w celu ich przetworzenia. Komponenty bean sterowane komunikatami mogą także nawiązać więcej połączeń z systemami IBM MQ z poziomu ich metody **onMessage()** . Więcej informacji na temat sposobu, w jaki połączenia te mogą korzystać z automatycznego ponownego połączenia klienta, zawiera sekcja Enterprise JavaBeans and Web-based applications(Komponenty EJB i aplikacje 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 (Message-driven bean-MDB), które są zainstalowane w środowisku Java EE , takie jak WebSphere Application Server, mogą używać specyfikacji aktywowania do przetwarzania komunikatów w systemie IBM MQ .

Specyfikacje aktywowania służą do wykrywania komunikatów przychodzących do systemu IBM MQ , a następnie dostarczania ich do komponentów MDB w celu ich przetworzenia. W tej sekcji opisano sposób, w jaki specyfikacja aktywowania monitoruje system IBM MQ .

Systemy MPB mogą również nawiązać dodatkowe połączenia z systemami IBM MQ z poziomu ich metody `onMessage()` .

Szczegółowe informacje na temat tego, w jaki sposób te połączenia mogą korzystać z automatycznego ponownego połączenia klienta, można znaleźć w sekcji “Korporacyjne komponenty JavaBeans i aplikacje z interfejsem WWW” na stronie 294.

CONNECTIONNAMELIST property

Podczas uruchamiania specyfikacja aktywowania próbuje nawiązać połączenie z menedżerem kolejek przy użyciu następujących elementów:

- Jeden określony we właściwości **QMANAGER** .
- Kanał wymieniony we właściwości **CHANNEL**
- Informacje o nazwie hosta i porcie z pierwszej pozycji w **CONNECTIONNAMELIST**

Jeśli specyfikacja aktywowania nie może połączyć się z menedżerem kolejek przy użyciu pierwszej pozycji na liście, specyfikacja aktywowania przechodzi do drugiej pozycji itd. dopóki nie zostanie nawiązane połączenie z menedżerem kolejek lub zostanie osiągnięty koniec listy.

Jeśli specyfikacja aktywowania nie może nawiązać połączenia z określonym menedżerem kolejek, za pomocą dowolnego z wpisów w **CONNECTIONNAMELIST**, specyfikacja aktywowania zostanie zatrzymana i musi zostać zrestartowana.

Po uruchomieniu specyfikacji aktywowania specyfikacja aktywowania pobiera komunikaty z systemu IBM MQ i dostarcza komunikaty do komponentu MDB w celu przetworzenia.

Jeśli menedżer kolejek zakończy się niepowodzeniem podczas przetwarzania komunikatu, środowisko Java EE wykryje awarię i podejmie próbę ponownego połączenia specyfikacji aktywowania.

Specyfikacja aktywowania korzysta z informacji znajdujących się we właściwości **CONNECTIONNAMELIST** , jak wcześniej, gdy specyfikacja aktywowania wykonuje próby ponownego połączenia.

If the activation specification tries all of the entries in the **CONNECTIONNAMELIST** and is still unable to connect to the queue manager, then the activation specification waits for the period of time specified by the IBM MQ resource adapter property **reconnectionRetryInterval** before trying again.

The IBM MQ resource adapter property **reconnectionRetryCount** defines the number of consecutive reconnection attempts that are be made before an activation specification is stopped,, and requires a manual restart

Gdy specyfikacja aktywowania ponownie nawiąże połączenie z systemem IBM MQ , środowisko Java EE wykonuje wszelkie wymagane procedury czyszczące transakcyjne oraz wznawia dostarczanie komunikatów do komponentów MDB w celu ich przetworzenia.

Aby procedura czyszcząca transakcji mogła działać poprawnie, środowisko Java EE musi mieć dostęp do dzienników dla menedżera kolejek, który uległ awarii.

Jeśli specyfikacje aktywowania są używane z transakcyjnymi bazami danych MDB, które biorą udział w transakcjach XA, i nawiązują połączenie z menedżerem kolejek z wieloma instancjami, **CONNECTIONNAMELIST** musi zawierać wpis dla instancji aktywnej i rezerwowej menedżera kolejek.

Oznacza to, że środowisko produktu Java EE może uzyskać dostęp do dzienników menedżera kolejek, jeśli środowisko musi przeprowadzić odtwarzanie transakcji, niezależnie od tego, który menedżer kolejek nawiązuje połączenie z następującymi po sobie awariami.

Jeśli transakcyjne komponenty MDB są używane z autonomicznymi menedżerami kolejek, właściwość **CONNECTIONNAMELIST** musi zawierać pojedynczą pozycję, aby upewnić się, że specyfikacja aktywowania zawsze ponownie łączy się z tym samym menedżerem kolejek działającym w tym samym systemie po awarii.

CCDTURL property

Po uruchomieniu specyfikacja aktywowania próbuje połączyć się z menedżerem kolejek określonym we właściwości **QMANAGER** przy użyciu pierwszej pozycji w tabeli definicji kanału klienta (CCDT).

Jeśli specyfikacja aktywowania nie może połączyć się z menedżerem kolejek przy użyciu pierwszej pozycji w tabeli, specyfikacja aktywowania przechodzi do drugiej pozycji itd. dopóki nie zostanie nawiązane połączenie z menedżerem kolejek lub zostanie osiągnięty koniec tabeli.

Jeśli specyfikacja aktywowania nie może nawiązać połączenia z określonym menedżerem kolejek przy użyciu któregokolwiek z wpisów w tabeli definicji kanału klienta, specyfikacja aktywowania zostanie zatrzymana i musi zostać zrestartowana.

Po uruchomieniu specyfikacji aktywowania specyfikacja aktywowania pobiera komunikaty z systemu IBM MQ i dostarcza komunikaty do komponentu MDB w celu przetworzenia.

Jeśli menedżer kolejek zakończy się niepowodzeniem podczas przetwarzania komunikatu, środowisko Java EE wykryje awarię i podejmie próbę ponownego połączenia specyfikacji aktywowania.

Specyfikacja aktywowania korzysta z informacji znajdujących się we właściwości CCDT, jak wcześniej, gdy specyfikacja aktywowania wykonuje próby ponownego połączenia.

If the activation specification tries all of the entries in the CCDT and is still unable to connect to the queue manager, the activation specification waits for the period of time specified by the IBM MQ resource adapter property **reconnectionRetryInterval** before trying again.

The IBM MQ resource adapter property **reconnectionRetryCount** defines the number of consecutive reconnection attempts that are be made before an activation specification is stopped,, and requires a manual restart

Gdy specyfikacja aktywowania ponownie nawiąże połączenie z systemem IBM MQ , środowisko Java EE wykonuje wszelkie wymagane procedury czyszczące transakcyjne oraz wznawia dostarczanie komunikatów do komponentów MDB w celu ich przetworzenia.

Aby procedura czyszcząca transakcji mogła działać poprawnie, środowisko Java EE musi mieć dostęp do dzienników dla menedżera kolejek, który uległ awarii.

Jeśli specyfikacje aktywowania są używane z transakcyjnymi bazami danych MDB, które biorą udział w transakcjach XA, i nawiązują połączenie z menedżerem kolejek z wieloma instancjami, pakiet CCDT musi zawierać wpis 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 musi przeprowadzić odtwarzanie transakcji, niezależnie od tego, który menedżer kolejek nawiązuje połączenie z następującymi po sobie awariami.

Jeśli transakcyjne komponenty MDB są używane z autonomicznymi menedżerami kolejek, pakiet CCDT musi zawierać pojedynczą pozycję, aby upewnić się, że specyfikacja aktywowania zawsze łączy się ponownie z tym samym menedżerem kolejek działającym w tym samym systemie po wystąpieniu awarii.

Upewnij się, że dla właściwości **AFFINITY** w CCDTs została ustawiona wartość domyślna *PREFERRED* , która jest używana ze specyfikacjami aktywowania, tak aby połączenia były nawiązane z tym samym aktywnym menedżerem kolejek.

CLIENTRECONNECTOPTIONS property

Specyfikacje aktywowania udostępniają własne funkcje ponownego połączenia. Udostępnione funkcje umożliwiają automatyczne ponowne nawiązanie połączenia z systemem IBM MQ , jeśli menedżer kolejek, z którym nawiązano połączenie, nie powiedzie się.

Z tego powodu funkcja automatycznego ponownego połączenia klienta udostępniana przez produkt IBM MQ classes for JMS nie jest obsługiwana.

Należy ustawić właściwość **CLIENTRECONNECTOPTIONS** na wartość *DISABLED* (wyłączone) dla wszystkich specyfikacji aktywowania, które są używane w Java EE.

Porty nasłuchiwania produktu WebSphere Application Server
Aplikacje komponentu bean sterowanego komunikatami (MDB), które są instalowane 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 ich przetworzenia. W tym temacie wyjaśniono, w jaki sposób port nasłuchiwania monitoruje system IBM MQ .

Systemy MPB mogą również nawiązać dodatkowe połączenia z systemami IBM MQ z poziomu ich metody `onMessage()` .

Więcej informacji na temat tego, w jaki sposób połączenia mogą korzystać z automatycznego ponownego połączenia klienta, zawiera sekcja "Korporacyjne komponenty JavaBeans i aplikacje z interfejsem WWW" na stronie 294 .

W przypadku portów nasłuchiwania produktu WebSphere Application Server :

- Produkty **CONNECTIONNAMELIST** i **CCDTURL** są obsługiwane z ograniczeniami
- **CLIENTRECONNECTOPTIONS** nie jest obsługiwany

CONNECTIONNAMELIST property

Porty nasłuchiwania korzystają z pul połączeń produktu JMS podczas nawiązywania połączenia z produktem IBM MQ, dlatego są one związane z konsekwencjami korzystania z pul połączeń. Więcej informacji na ten temat zawiera sekcja "Specyfikacje aktywowania" na stronie 289.

Jeśli nie ma wolnych połączeń, a maksymalna liczba połączeń nie została jeszcze utworzona z tej fabryki połączeń, właściwość **CONNECTIONNAMELIST** jest używana do próby utworzenia nowego połączenia z produktem IBM MQ.

Jeśli wszystkie systemy IBM MQ w **CONNECTIONNAMELIST** nie są dostępne, port nasłuchiwania zostanie zatrzymany.

Następnie port nasłuchiwania czeka przez okres określony przez właściwość niestandardową usługi nasłuchiwania komunikatów **RECOVERY.RETRY.INTERVAL** i próbuje ponownie nawiązać połączenie.

Ta próba ponownego nawiązania połączenia sprawdza, czy w puli połączeń istnieją jakiekolwiek wolne połączenia, tylko w przypadku, gdy jeden z nich został zwrócony między próbami połączenia. Jeśli jeden z nich nie jest dostępny, port nasłuchiwania korzysta z **CONNECTIONNAMELIST** jak wcześniej.

Po ponownym nawiązaniu połączenia przez port nasłuchiwania z systemem IBM MQ środowisko Java EE wykonuje wszelkie wymagane procedury czyszczące transakcyjne, a następnie wznawia dostarczanie komunikatów do komponentów MDB w celu ich przetworzenia.

Aby procedura czyszcząca transakcji mogła działać poprawnie, środowisko Java EE musi mieć dostęp do dzienników dla menedżera kolejek, który uległ awarii.

Jeśli porty nasłuchiwania są używane z transakcyjnymi bazami danych MDB, które biorą udział w transakcjach XA, i nawiązują połączenie z **menedżerem kolejek z wieloma instancjami**, **CONNECTIONNAMELIST** musi zawierać wpis dla aktywnej i rezerwowej instancji menedżera kolejek.

Oznacza to, że środowisko produktu Java EE może uzyskać dostęp do dzienników menedżera kolejek, jeśli środowisko musi przeprowadzić odtwarzanie transakcji, niezależnie od tego, który menedżer kolejek nawiązuje połączenie z następującymi po sobie awariami.

Jeśli transakcyjne komponenty MDB są używane z autonomicznymi menedżerami kolejek, właściwość **CONNECTIONNAMELIST** musi zawierać pojedynczą pozycję, aby upewnić się, że specyfikacja aktywowania zawsze ponownie łączy się z tym samym menedżerem kolejek działającym w tym samym systemie po awarii.

CCDTURL property

Podczas uruchamiania port nasłuchiwania próbuje połączyć się z menedżerem kolejek określonym we właściwości **QMANAGER** przy użyciu pierwszej pozycji w tabeli definicji kanału klienta.

Jeśli port nasłuchiwania nie jest w stanie nawiązać połączenia z menedżerem kolejek przy użyciu pierwszej pozycji w tabeli, port nasłuchiwania zostanie przeniesiony do drugiej pozycji i tak dalej, dopóki nie zostanie nawiązane połączenie z menedżerem kolejek lub zostanie osiągnięty koniec tabeli.

Jeśli port nasłuchiwania nie może połączyć się z określonym menedżerem kolejek przy użyciu któregokolwiek z wpisów w tabeli definicji kanału klienta, port nasłuchiwania zostanie zatrzymany.

Następnie port nasłuchiwania czeka przez okres określony przez właściwość niestandardową usługi nasłuchiwania komunikatów **RECOVERY.RETRY.INTERVAL** i próbuje ponownie nawiązać połączenie.

Ta próba ponownego połączenia działa w ten sposób przez wszystkie pozycje w tabeli definicji kanału klienta, jak wcześniej.

Gdy port nasłuchiwania jest uruchomiony, pobiera komunikaty z systemu IBM MQ i dostarcza je do komponentu MDB w celu przetworzenia.

Jeśli menedżer kolejek zakończy się niepowodzeniem podczas przetwarzania komunikatu, środowisko Java EE wykryje awarię i podejmie próbę ponownego połączenia portu nasłuchiwania. Port nasłuchiwania korzysta z informacji znajdujących się w tabeli definicji kanału klienta podczas wykonywania prób ponownego połączenia.

Jeśli port nasłuchiwania podejmie próbę wszystkich pozycji w tabeli definicji kanału klienta i nadal nie może nawiązać połączenia z menedżerem kolejek, to przed ponowną próbą portu oczekuje na upływ czasu określonego przez właściwość **RECOVERY.RETRY.INTERVAL** .

Właściwość usługi nasłuchiwania komunikatów **MAX.RECOVERY.RETRIES** definiuje liczbę kolejnych prób ponownego połączenia, które są wykonywane przed zatrzymaniem portu nasłuchiwania i wymaga ręcznego restartu.

Po ponownym nawiązaniu połączenia przez port nasłuchiwania z systemem IBM MQ środowisko Java EE wykonuje wszelkie wymagane procedury czyszczące transakcyjne, a następnie wznawia dostarczanie komunikatów do komponentów MDB w celu ich przetworzenia.

Aby procedura czyszcząca transakcji mogła działać poprawnie, środowisko Java EE musi mieć dostęp do dzienników dla menedżera kolejek, który uległ awarii.

Jeśli porty nasłuchiwania są używane z transakcyjnymi bazami danych MDB, które biorą udział w transakcjach XA, i nawiązują połączenie z menedżerem kolejek z wieloma instancjami, pakiet CCDT musi zawierać wpis 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 musi przeprowadzić odtwarzanie transakcji, niezależnie od tego, który menedżer kolejek nawiązuje połączenie z następującymi po sobie awariami.

Jeśli transakcyjne komponenty MDB są używane z autonomicznymi menedżerami kolejek, pakiet CCDT musi zawierać pojedynczą pozycję, aby zapewnić, że port nasłuchiwania ponownie łączy się z tym samym menedżerem kolejek działającym w tym samym systemie po wystąpieniu awarii.

Należy upewnić się, że ustawiono wartość domyślną *PREFERRED* dla właściwości **AFFINITY** w CCDTs, która jest używana z portami nasłuchiwania, tak aby połączenia były nawiązane z tym samym aktywnym menedżerem kolejek.

CLIENTRECONNECTOPTIONS property

Porty nasłuchiwania udostępniają własne funkcje ponownego połączenia. Udostępnione funkcje umożliwiają portale nasłuchiwania automatyczne ponowne nawiąwanie połączenia z systemem IBM MQ , jeśli menedżer kolejek, z którym nawiązano połączenie, nie powiedzie się.

Z tego powodu funkcja automatycznego ponownego połączenia klienta udostępniana przez produkt IBM MQ classes for JMS nie jest obsługiwana.

Należy ustawić właściwość **CLIENTRECONNECTOPTIONS** na wartość *DISABLED* dla wszystkich portów nasłuchiwania, które są używane w Java EE.

Korporacyjne komponenty JavaBeans i aplikacje z interfejsem WWW
Aplikacje i aplikacje Enterprise JavaBean (EJB), które działają w obrębie kontenera WWW, takie jak serwlety, używają fabryki połączeń produktu JMS do tworzenia połączenia z menedżerem kolejek produktu IBM MQ .

Następujące ograniczenia mają zastosowanie do komponentów EJB i aplikacji z interfejsem WWW:

- Produkty **CONNECTIONNAMELIST** i **CCDTURL** są obsługiwane z ograniczeniami
- **CLIENTRECONNECTOPTIONS** nie jest obsługiwany

CONNECTIONNAMELIST property

Jeśli środowisko Java EE udostępnia pulę połączeń dla połączeń JMS, należy zapoznać się z informacjami na temat wpływu tej właściwości na zachowanie właściwości **CONNECTIONNAMELIST** , patrz sekcja “Używanie opcji CONNECTIONNAMELIST lub CCDT w puli połączeń” na stronie 295 .

Jeśli środowisko Java EE nie udostępnia puli połączeń produktu JMS . aplikacja korzysta z właściwości **CONNECTIONNAMELIST** w taki sam sposób, jak aplikacje produktu Java SE .

Jeśli aplikacje są używane z transakcyjnymi bazami danych MDB, które biorą udział w transakcjach XA i nawiązują połączenie z menedżerem kolejek z wieloma instancjami, **CONNECTIONNAMELIST** musi zawierać wpis dla instancji aktywnej i rezerwowej menedżera kolejek.

Oznacza to, że środowisko produktu Java EE może uzyskać dostęp do dzienników menedżera kolejek, jeśli środowisko musi przeprowadzić odtwarzanie transakcji, niezależnie od tego, który menedżer kolejek nawiązuje połączenie z następującymi po sobie awariami.

Jeśli aplikacje są używane z autonomicznymi menedżerami kolejek, właściwość **CONNECTIONNAMELIST** musi zawierać pojedynczą pozycję, aby upewnić się, że aplikacja zawsze ponownie łączy się z tym samym menedżerem kolejek, działającym w tym samym systemie, po wystąpieniu awarii.

CCDTURL property

Jeśli środowisko Java EE udostępnia pulę połączeń dla połączeń produktu JMS , należy zapoznać się z informacjami na temat sposobu, w jaki wpływa to na zachowanie właściwości **CCDTURL** , zawiera sekcja “Używanie opcji CONNECTIONNAMELIST lub CCDT w puli połączeń” na stronie 295 .

Jeśli środowisko Java EE nie udostępnia puli połączeń produktu JMS . aplikacja korzysta z właściwości **CCDTURL** w taki sam sposób, jak aplikacje produktu Java SE .

Jeśli aplikacje są używane z transakcyjnymi bazami danych MDB, które biorą udział w transakcjach XA, i nawiązują połączenie z menedżerem kolejek z wieloma instancjami, pakiet CCDT musi zawierać wpis 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 musi przeprowadzić odtwarzanie transakcji, niezależnie od tego, który menedżer kolejek nawiązuje połączenie z następującymi po sobie awariami.

Jeśli aplikacje są używane z autonomicznymi menedżerami kolejek, pakiet CCDT musi zawierać pojedynczą pozycję, aby upewnić się, że specyfikacja aktywowania zawsze ponownie łączy się z tym samym menedżerem kolejek działającym w tym samym systemie po wystąpieniu awarii.

CLIENTRECONNECTOPTIONS property

Należy ustawić właściwość **CLIENTRECONNECTOPTIONS** na wartość *DISABLED* dla wszystkich fabryk połączeń produktu JMS używanych przez komponenty EJB lub aplikacje, które działają w kontenerze WWW.

Aplikacje, które wymagają automatycznego ponownego nawiązania połączenia z nowym menedżerem kolejek, jeśli używany menedżer kolejek nie powiedzie się, należy zaimplementować własną logikę

ponownego połączenia. Więcej informacji można znaleźć w sekcji “Implementowanie logiki ponownego połączenia w aplikacji Java EE” na stronie 296.

Scenariusze: WebSphere Application Server z IBM MQ

Scenariusze: profil WebSphere Application Server Liberty z produktem IBM MQ

Aplikacje działające w kontenerach klienta
W niektórych środowiskach Java EE , takich jak WebSphere Application Server, należy udostępnić kontener klienta, który może być używany do uruchamiania aplikacji produktu Java SE .

Aplikacje działające w tych środowiskach korzystają z fabryki połączeń produktu JMS w celu nawiązania połączenia z menedżerem kolejek produktu IBM MQ .

Dla aplikacji działających w kontenerach klienta:

- Produkty **CONNECTIONNAMELIST** i **CCDTURL** są w pełni obsługiwane.
- **CLIENTRECONNECTOPTIONS** jest w pełni obsługiwany

CONNECTIONNAMELIST property

Jeśli środowisko Java EE udostępnia pulę połączeń dla połączeń JMS, należy zapoznać się z informacjami na temat wpływu tej właściwości na zachowanie właściwości **CONNECTIONNAMELIST** , patrz sekcja “Używanie opcji CONNECTIONNAMELIST lub CCDT w puli połączeń” na stronie 295 .

Jeśli środowisko Java EE nie udostępnia puli połączeń produktu JMS . aplikacja korzysta z właściwości **CONNECTIONNAMELIST** w taki sam sposób, jak aplikacje produktu Java SE .

CCDTURL property

Jeśli środowisko Java EE udostępnia pulę połączeń dla połączeń produktu JMS , należy zapoznać się z informacjami na temat sposobu, w jaki wpływa to na zachowanie właściwości **CCDTURL** , zawiera sekcja “Używanie opcji CONNECTIONNAMELIST lub CCDT w puli połączeń” na stronie 295 .

Jeśli środowisko Java EE nie udostępnia puli połączeń produktu JMS . aplikacja korzysta z właściwości **CCDTURL** w taki sam sposób, jak aplikacje produktu Java SE .

Używanie opcji CONNECTIONNAMELIST lub CCDT w puli połączeń
W niektórych środowiskach Java EE , na przykład WebSphere Application Server, podaj pulę połączeń JMS . Kontener, który może być używany do uruchamiania aplikacji produktu Java SE .

Aplikacje, które tworzą połączenie przy użyciu fabryki połączeń, która została zdefiniowana w środowisku Java EE , albo uzyskują istniejące wolne połączenie z puli połączeń dla tej fabryki połączeń, albo nowe połączenie, jeśli w puli połączeń nie ma odpowiedniego połączenia.

Może to mieć wpływ na to, czy fabryka połączeń została skonfigurowana przy użyciu zdefiniowanej właściwości **CONNECTIONNAMELIST** lub **CCDTURL** .

Przy pierwszym użyciu fabryki połączeń w celu utworzenia połączenia środowisko Java EE korzysta z produktu **CONNECTIONNAMELIST**. lub **CCDTURL** , aby utworzyć nowe połączenie z systemem IBM MQ . Jeśli to połączenie nie jest już wymagane, jest ono zwracane do puli połączeń, w której połączenie staje się dostępne do ponownego wykorzystania.

Jeśli inne połączenie tworzy połączenie z fabryki połączeń, środowisko Java EE zwraca połączenie z puli połączeń, a nie za pomocą właściwości **CONNECTIONNAMELIST** lub **CCDTURL** w celu utworzenia nowego połączenia.

Jeśli połączenie jest używane, gdy instancja menedżera kolejek nie powiedzie się, połączenie zostanie usunięte. Jednak treść puli połączeń może nie być, co oznacza, że pula może potencjalnie zawierać połączenia z menedżerem kolejek, który nie jest już uruchomiony.

W takiej sytuacji, gdy zostanie wykonane żądanie utworzenia połączenia z fabryki połączeń, zostanie zwrócone połączenie z menedżerem kolejek zakończonych niepowodzeniem. Wszelkie próby użycia tego

połączenia nie powiodą się, ponieważ menedżer kolejek nie jest już uruchomiony, co powoduje usunięcie połączenia.

Tylko wtedy, gdy pula połączeń jest pusta, środowisko Java EE będzie używać właściwości **CONNECTIONNAMELIST** lub **CCDTURL** w celu utworzenia nowego połączenia z produktem IBM MQ.

Ze względu na sposób, w jaki produkty **CONNECTIONNAMELIST** i CCDTs są używane do tworzenia połączeń JMS , możliwe jest również utworzenie puli połączeń zawierającej połączenia z różnymi systemami IBM MQ .

Na przykład załóżmy, że fabryka połączeń została skonfigurowana z właściwością **CONNECTIONNAMELIST** ustawioną na następującą wartość:

```
CONNECTIONNAMELIST = hostname1(port1), hostname2(port2)
```

Załóżmy, że po raz pierwszy aplikacja próbuje utworzyć połączenie z autonomicznym menedżerem kolejek z tej fabryki połączeń, a menedżer kolejek działający w systemie `hostname1(port1)` nie jest dostępny. Oznacza to, że aplikacja kończy się połączeniem z menedżerem kolejek działającym w systemie `hostname2(port2)`.

Kolejna aplikacja jest teraz dostarczana razem i tworzy połączenie JMS z tej samej fabryki połączeń. Menedżer kolejek w systemie `hostname1(port1)` jest teraz dostępny, dlatego nowe połączenie JMS jest tworzone dla tego systemu IBM MQ i jest zwracane do aplikacji.

Po zakończeniu obu aplikacji zamykają one połączenia programu JMS , co powoduje, że połączenia są zwracane do puli połączeń.

W wyniku tego pula połączeń dla fabryki połączeń zawiera teraz dwa połączenia produktu 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 być w stanie połączyć się 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 być automatycznie ponownie połączone, jeśli menedżer kolejek nie powiedzie się, muszą implementować własną logikę ponownego połączenia.

Poniższe opcje dają więcej informacji na temat tego, w jaki sposób można to osiągnąć.

Pozwól, aby aplikacja nie powiodła się

Takie podejście nie wymaga 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 usterkę. Należy zauważyć, że jest to również wymagane w przypadku niepowodzeń, takich jak MQRC_Q_FULL, które nie są powiązane 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 poprzedzającym kodzie założono, że fabryka połączeń, ten serwlet jest używany, ma zdefiniowaną właściwość **CONNECTIONNAMELIST** .

Gdy serwlet najpierw przetwarza, nowe połączenie jest tworzone przy użyciu właściwości **CONNECTIONNAMELIST** , przy założeniu, że żadne połączenia z puli nie są dostępne z innych aplikacji łączących się z tym samym menedżerem kolejek.

Po zwolniu połączenia po wywołaniu programu `close()` połączenie to jest zwracane do puli i ponownie wykorzystywane przy następnym uruchomieniu serwletu-bez odwołania się do serwera **CONNECTIONNAMELIST** -do momentu wystąpienia błędu połączenia, w którym to momencie generowane jest zdarzenie CONNECTION_ERROR_OCCURRED. To zdarzenie powoduje, że pula nie może zniszczyć połączenia, które się nie powiodło.

Po następnym uruchomieniu aplikacji żadne połączenie z puli nie jest dostępne, a program **CONNECTIONNAMELIST** jest używany do łączenia się z pierwszym dostępnym menedżerem kolejek. Jeśli wystąpiło niepowodzenie menedżera kolejek (na przykład awaria nie była awarią sieci przejściowej), serwlet łączy się z instancją kopii zapasowej, gdy jest ona dostępna.

Jeśli w aplikacji są zaangażowane inne zasoby, takie jak bazy danych, może być konieczne wskazanie, że serwer aplikacji powinien wycofać transakcję.

Obsługa ponownego połączenia w aplikacji

Jeśli program wywołujący nie może przetworzyć niepowodzenia z serwletu, ponowne nawiązanie połączenia musi zostać obsłużone w aplikacji. Jak pokazano w poniższym przykładzie, aby można było obsłużyć ponowne połączenie w aplikacji, aplikacja musi zażądać nowego połączenia, aby mógł buforować fabrykę połączeń, która została wyszukana z poziomu produktu JNDI , i obsługiwać `JMSException` , takie jak JMSCMQ0001: `Wywołanie funkcji WebSphere MQ nie powiodło się. Przyczyna: '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 zawiera pętlę uśpienia i ponowienia w celu obsługi ponownego połączenia bez natychmiastowego połączenia. W praktyce ta metoda zapobiega wielu próśb ponownego połączenia. Należy zauważyć, że warunki wyjścia zostały pominięte w przykładzie dla jasności.

```
 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 aplikacja zwalniała wszystkie połączenia, które są przechowywane w innych zasobach, niezależnie od tego, czy są to inne menedżery kolejek produktu IBM MQ , czy inne usługi zaplecza, takie jak bazy danych. Należy ponownie ustanowić te połączenia, gdy ponowne nawiązanie połączenia z nową instancją menedżera kolejek produktu IBM MQ jest zakończone. Jeśli połączenia nie zostaną ponownie nawiązane, zasoby serwera aplikacji będą niepotrzebnie wstrzymane podczas próby ponownego nawiązania połączenia, a limit czasu mógł zostać przekroczony o czasie, w którym są ponownie wykorzystywane.

Korzystanie z programu WorkManager

W przypadku długotrwałych aplikacji (na przykład przetwarzania wsadowego), w których czas przetwarzania jest dłuższy niż kilkadziesiąt sekund, można użyć programu WebSphere Application Server WorkManager . Przykład fragmentu kodu dla WebSphere Application Server jest następujący:

```
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 zaimplementowane za pomocą 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, powolna sieć lub rozległy dostęp do bazy danych (szczególnie w połączeniu z powolnym awariami), serwer zaczyna wyprowadzać zawieszone ostrzeżenia wątku, podobnie jak w następującym przykładzie:

WSVR0605W: Wątek "WorkManager.DefaultWorkManager : 0" (00000035) jest aktywny dla 694061 milisekund i może być zawieszony. Łączna liczba wątków na serwerze, które mogą być zawieszone, wynosi 1 wątek.

Ostrzeżenia te można zminimalizować, zmniejszając wielkość zadania wsadowego lub zwiększając limit czasu zawieszonego wątku. Jednak ogólnie zaleca się zaimplementowanie tego przetwarzania w komponencie EJB (w przypadku wysyłania wsadowego) lub komponentu bean sterowanego komunikatami (do przetwarzania lub odbierania lub odbierania odpowiedzi).

Należy zauważyć, że ponowne połączenie zarządzane przez aplikację nie udostępnia ogólnego rozwiązania w celu obsługi błędów w czasie wykonywania, a aplikacja musi nadal obsługiwać błędy, które nie są powiązane z awarią połączenia.

Na przykład próba umieszczenia komunikatu w kolejce, która jest pełna (2053 MQRC_Q_FULL), lub próba nawiązania połączenia z menedżerem kolejek przy użyciu niepoprawnych informacji autoryzacyjnych (2035 MQRC_NOT_AUTHORIZED).

Aplikacja musi także obsługiwać błędy 2059 MQRC_Q_MGR_NOT_AVAILABLE, gdy żadne instancje nie są natychmiast dostępne, gdy trwa przełączanie awaryjne. Może to zostać osiągnięte przez aplikację zgłaszając wyjątki JMS w miarę ich występowania, zamiast cichego próby ponownego nawiązania połączenia.

Zestawianie obiektów IBM MQ classes for JMS
Korzystanie z zestawiania połączeń poza produktem Java EE pomaga zmniejszyć ogólne obciążenie wynikające np. z niektórych autonomicznych aplikacji korzystających z frameworków lub wdrażanych

w środowiskach chmurowych, a także z większej liczby połączeń klienckich z produktem `QueueManagers` , co prowadzi do wzrostu konsolidacji serwerów aplikacji i menedżerów kolejek.

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, które są dostępne w ramach serwerów Java EE , odpowiadają różnym modelom programistycznym.

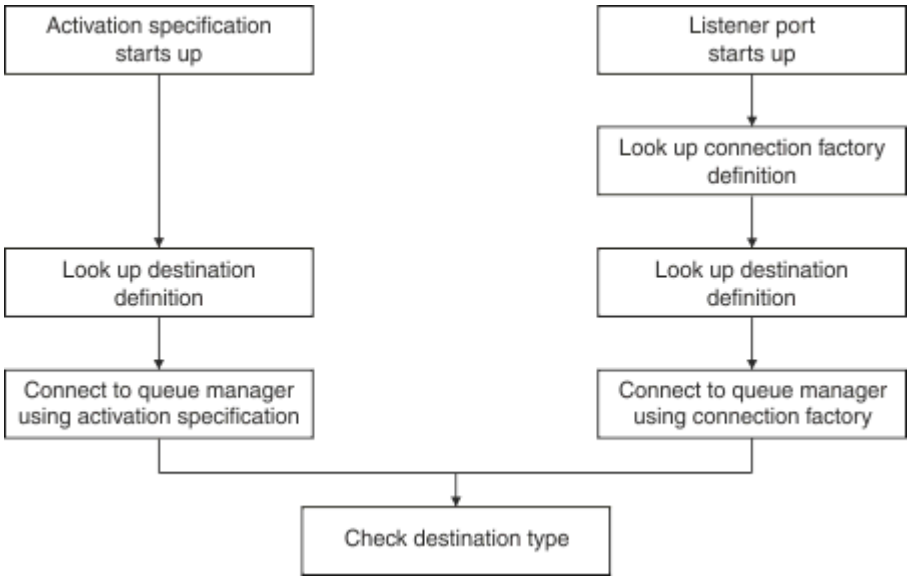
W przypadku produktu Java SE (lub z innymi ramami, takimi jak Spring) modele programistyczne są niezwykle elastyczne. Dlatego też jedna strategia łączenia nie pasuje do wszystkich. Należy rozważyć, czy istnieją ramy w miejscu, które mogłyby zrobić dowolną formę łączenia, na przykład, Wiosna.

Strategia zestawiania, która ma być używana, zależy od środowiska, w którym działa aplikacja.

Zestawianie obiektów w środowisku Java EE
Serwery aplikacji Java EE udostępniają funkcje zestawiania połączeń, które mogą być używane przez aplikacje bean sterowane komunikatami, komponenty Enterprise Java Bean i serwlety.

Produkt WebSphere Application Server obsługuje pulę połączeń z dostawcą JMS w celu zwiększenia wydajności. Gdy aplikacja tworzy połączenie z produktem JMS , serwer aplikacji określa, czy połączenie istnieje już w puli wolnych połączeń. Jeśli tak, to połączenie zostanie zwrócone do aplikacji.
W przeciwnym razie zostanie utworzone nowe połączenie.

Rysunek 47 na stronie 300 pokazuje, w jaki sposób zarówno specyfikacje aktywowania, jak i porty nasłuchiwania nawiązują połączenie JMS i używają tego połączenia do monitorowania miejsca docelowego komunikatów w trybie normalnym.



Rysunek 47. Tryb standardowy

Jeśli używany jest dostawca przesyłania komunikatów produktu IBM MQ , aplikacje, które wykonują komunikaty wychodzące (takie jak komponenty EJB i serwlety) oraz komponent portu nasłuchiwania komponentów bean sterowanych komunikatami, mogą korzystać z tych pul połączeń.

Specyfikacje aktywowania dostawcy przesyłania komunikatów produktu IBM MQ korzystają z funkcji zestawiania połączeń udostępnianej przez adapter zasobów produktu IBM MQ . Więcej informacji na ten temat zawiera sekcja Konfigurowanie właściwości dla adaptera zasobów produktu WebSphere MQ .

W produkcie “Przykłady korzystania z puli połączeń” na stronie 304 wyjaśniono, w jaki sposób aplikacje, które wykonują komunikaty wychodzące, oraz porty nasłuchiwania, korzystają z puli wolnych podczas tworzenia połączeń produktu JMS .

“Wątki konserwacji puli połączeń” na stronie 307 wyjaśnia, co dzieje się z tymi połączeniami, gdy aplikacja lub port nasłuchiwania zakończył się połączeniami.

“Przykłady wątków konserwacji puli” na stronie 309 wyjaśnia, w jaki sposób pula połączeń wolnych jest czyszczona, aby zapobiec stałowaniu się połączeń produktu JMS .

Produkt WebSphere Application Server ma limit liczby połączeń, które można utworzyć z fabryki, określonej przez właściwość *Maksymalna liczba połączeń* fabryki połączeń. Wartością domyślną tej właściwości jest 10, co oznacza, że w danym momencie z fabryki może być utworzonych maksymalnie 10 połączeń.

Każda fabryka ma powiązaną wolną pulę połączeń. Gdy serwer aplikacji zostanie uruchomiony, wolne pule połączeń są puste. Maksymalna liczba połączeń, które mogą istnieć w wolnej puli dla fabryki, jest również określona przez właściwość Maksymalna liczba połączeń.

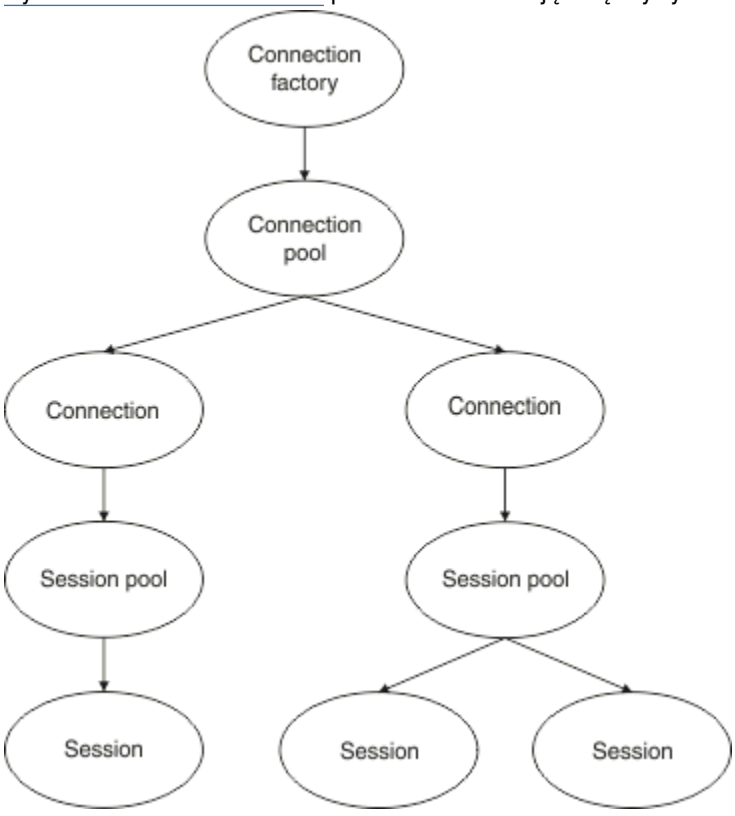
Wskazówka: Za pomocą produktu JMS 2.0można użyć fabryki połączeń w celu utworzenia obu połączeń i kontekstów. Dzięki temu możliwe jest, że istnieje pula połączeń powiązana z fabryką połączeń, która zawiera połączenie obu połączeń i kontekstów. Zaleca się, aby fabryka połączeń była używana tylko do tworzenia połączeń lub tworzenia kontekstów. Dzięki temu pula połączeń dla tej fabryki połączeń będzie zawierała tylko obiekty pojedynczego typu, co sprawia, że pula jest bardziej wydajna.

Więcej informacji na temat sposobu łączenia połączeń w produkcie WebSphere Application Serverzawiera sekcja Konfigurowanie zestawiania połączeń dla połączeń JMS. W przypadku innych serwerów aplikacji należy zapoznać się z dokumentacją odpowiedniego serwera aplikacji.

Sposób użycia puli połączeń

Każda fabryka połączeń produktu JMS ma powiązaną pulę połączeń, a pula połączeń zawiera zero lub większą liczbę połączeń produktu JMS . Każde połączenie JMS ma powiązaną pulę sesji JMS , a każda pula sesji JMS zawiera zero lub więcej sesji JMS .

Rysunek 48 na stronie 301 przedstawia relację między tymi obiektami.



Rysunek 48. Pule połączeń i pule sesji

Gdy port nasłuchiwania zostanie uruchomiony lub aplikacja, która chce wykonać wychodzące przesyłanie komunikatów, korzysta z fabryki w celu 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 produktu WebSphere Application Server próbuje uzyskać połączenie z wolnej puli dla tej fabryki i zwrócić ją do aplikacji.

Jeśli w puli nie ma żadnych 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 dla aplikacji, która ma być używana.

Jeśli jednak aplikacja próbuje utworzyć połączenie, ale liczba połączeń utworzonych z tej fabryki jest już równa właściwości *Maksymalna liczba połączeń* fabryki, aplikacja oczekuje na udostępnienie połączenia (w celu umieszczenia w puli wolnej).

Czas oczekiwania aplikacji jest określony we właściwości *limitu czasu połączenia* puli połączeń, która ma wartość domyślną 180 sekund. Jeśli połączenie zostanie ponownie umieszczone w puli wolnych pul w ciągu tego 180 sekund, menedżer połączeń natychmiast pobiera go z puli i przekazuje je do aplikacji. Jeśli jednak upłynie limit czasu limitu czasu, zostanie zgłoszony wyjątek *ConnectionWaitTimeoutException* .

Gdy aplikacja zakończy połączenie i zamknie ją, wywołując komendę:

- **Connection.close()**
- **QueueConnection.close()**
- **TopicConnection.close()**

Połączenie jest rzeczywiście otwarte i jest zwracane do puli wolnych, aby można było ponownie wykorzystać je przez inną aplikację. Z tego powodu istnieje możliwość otwarcia połączeń między produktem WebSphere Application Server a dostawcą JMS , nawet jeśli na serwerze aplikacji nie są uruchomione żadne aplikacje produktu JMS .

Zaawansowane właściwości puli połączeń
Istnieje wiele zaawansowanych właściwości, które mogą być używane do sterowania zachowaniem pul połączeń produktu JMS .

Ochrona przeciwprzepięciowa

W produkcie “W jaki sposób aplikacje, które wykonują komunikaty wychodzące, korzystają z puli połączeń” na stronie 306 opisano sposób użycia metody `sendMessage()`, która zawiera produkt `connectionFactory.createConnection()`.

Należy rozważyć sytuację, w której w przypadku 50 komponentów EJB wszystkie połączenia JMS są tworzone z tej samej fabryki połączeń, co część ich metody `ejbCreate()` .

Jeśli wszystkie te komponenty bean są tworzone w tym samym czasie i nie ma żadnych połączeń w puli połączeń wolnych od fabryki, serwer aplikacji próbuje jednocześnie utworzyć 50 JMS połączeń z tym samym dostawcą JMS . Wynikiem jest znaczące obciążenie zarówno dla WebSphere Application Server , jak i dla dostawcy JMS .

Właściwości ochrony przeciwprzepięciowej mogą zapobiec takiej sytuacji, ograniczając liczbę połączeń JMS , które mogą być tworzone z fabryki połączeń w dowolnym momencie, a także roztaczając tworzenie dodatkowych połączeń.

Ograniczanie liczby połączeń JMS w dowolnym momencie jest realizowane przy użyciu dwóch właściwości:

- Próg przeciągi

- Interwał tworzenia przeciąg.

Gdy aplikacje EJB próbują utworzyć połączenie z produktem 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 właściwość `surge threshold` , menedżer połączeń czeka przez okres określony przez właściwość `surge creation interval` przed utworzeniem nowego połączenia i otwarciem go.

Zablokowane połączenia

Połączenie JMS jest uznawane za `stuck`, jeśli aplikacja JMS używa tego połączenia do wysłania żądania do dostawcy JMS , a dostawca nie odpowie w określonym czasie.

Produkt WebSphere Application Server umożliwia wykrywanie połączeń `stuck` JMS w celu korzystania z tej funkcji, należy ustawić trzy właściwości:

- Zegar czasu zablokowanych połączeń
- Czas zablokowanych
- Próg zablokowania

“Przykłady wątków konserwacji puli” na stronie 309 wyjaśnia, w jaki sposób wątek konserwacji puli jest okresowo uruchamiany i sprawdza zawartość wolnej puli w fabryce połączeń, szukając połączeń, które nie były nieużywane przez pewien czas lub które były zbyt długo dostępne.

Aby wykryć zablokowane połączenia, serwer aplikacji zarządza również wątkiem połączenia zablokowanych połączeń, który sprawdza stan wszystkich aktywnych połączeń utworzonych z fabryki połączeń w celu sprawdzenia, czy którekolwiek z nich oczekuje na odpowiedź od dostawcy JMS .

Gdy wątek zablokowanych połączeń jest uruchamiany, jest on 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 jeden oczekujący na odpowiedź, określa czas oczekiwania na odpowiedź i porównuje ten czas z wartością właściwości `Stuck time` .

Jeśli czas odpowiedzi dostawcy JMS na odpowiedź przekracza czas określony przez właściwość `Stuck time` , serwer aplikacji oznacza połączenie JMS jako zablokowane.

Na przykład: przypuśćmy, że fabryka połączeń `jms/CF1` ma właściwość `Stuck time timer` ustawioną na wartość 10, a właściwość `Stuck time` ustawioną na wartość 15.

Wątek połączeń zablokowanych staje się aktywny co 10 sekund i sprawdza, czy jakiekolwiek połączenie utworzone z produktu `jms/CF1` oczekuje na odpowiedź od IBM MQdłużej niż 15 sekund.

Załóżmy, że komponent EJB tworzy połączenie produktu JMS z produktem IBM MQ przy użyciu produktu `jms/CF1`, a następnie próbuje utworzyć sesję produktu JMS przy użyciu tego połączenia, wywołując komendę `Connection.createSession()`.

Jednak dostawca produktu JMS nie może odpowiedzieć na żądanie. Być może komputer został zamrożony lub proces uruchomiony na dostawcy JMS jest zakleszczany, co uniemożliwienie przetwarzania nowej pracy:

Dziesięć sekund po nazwie komponentu EJB o nazwie `Connection.createSession()`, licznik czasu połączenia zablokowanych staje się aktywny i wygląda na aktywne połączenia utworzone z programu `jms/CF1`.

Załóżmy, że istnieje tylko jedno aktywne połączenie, na przykład o nazwie `c1`. Pierwszy komponent EJB czeka 10 sekund na odpowiedź na żądanie wysłane do produktu `c1`, które jest mniejsze niż wartość `Stuck time`, więc licznik czasu połączenia zablokowanych połączeń ignoruje to połączenie i staje się nieaktywny.

10 sekund później wątek połączenia zablokowanych staje się aktywny ponownie i wygląda na aktywne połączenia dla programu `jms/CF1`. Tak jak poprzednio, założono, że istnieje tylko jedno połączenie, `c1`.

Jest to teraz 20 sekund od pierwszego komponentu EJB o nazwie `createSession()`, a komponent EJB nadal oczekuje na odpowiedź. Wartość 20 sekund jest dłuższa niż czas określony we właściwości `Stuck time` , dlatego wątek połączenia jest oznaczony jako `c1` jako zablokowany.

Jeśli pięć sekund później program IBM MQ w końcu odpowie i zezwoli na utworzenie sesji programu JMS przez pierwszy komponent EJB, połączenie jest ponownie używane.

Serwer aplikacji zlicza liczbę połączeń JMS utworzonych z fabryki połączeń, które są zablokowane. Gdy aplikacja korzysta z tej fabryki połączeń w celu utworzenia nowego połączenia z serwerem JMS , a w puli wolnych tej fabryki nie ma żadnych 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`, to menedżer połączeń utworzy nowe połączenie i poda je do aplikacji.

Jeśli jednak liczba zablokowanych połączeń jest równa wartości właściwości `Stuck threshold` , aplikacja pobiera wyjątek zasobu.

Partycje puli

Produkt WebSphere Application Server udostępnia dwie właściwości, które umożliwiają partycjonowanie wolnej puli połączeń dla fabryki połączeń:

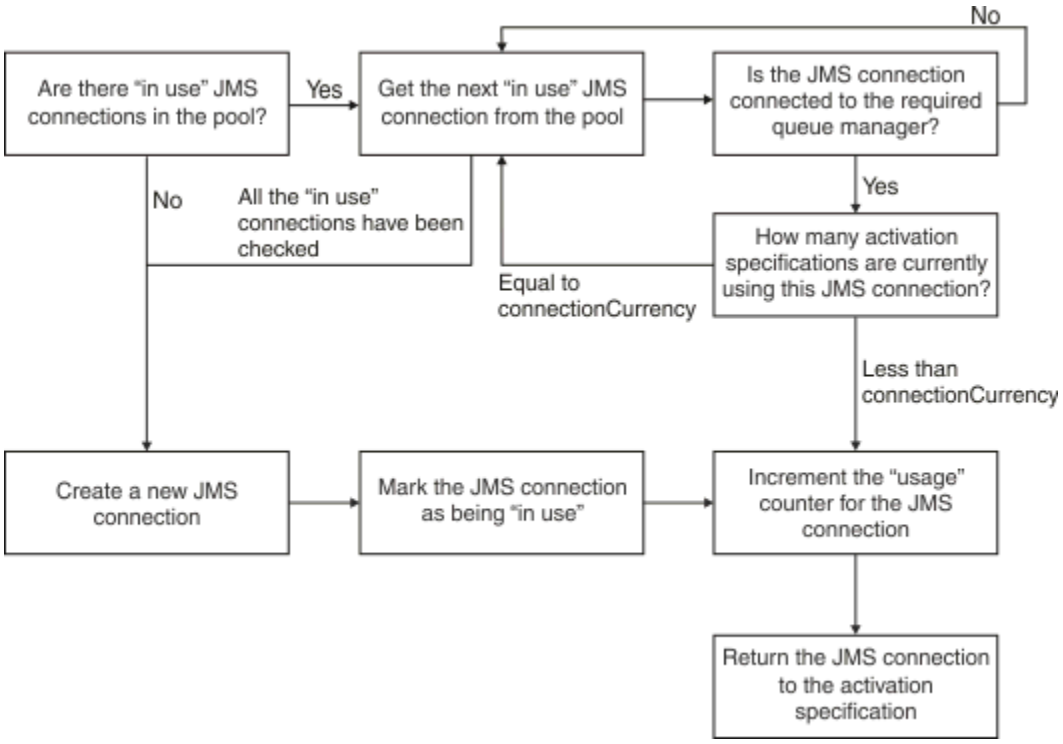
- Program `Number of free pool partitions` informuje serwer aplikacji o liczbie partycji, do których ma zostać podzielona wolna pula połączeń.
- `Free pool distribution table size` określa, w jaki sposób partycje są indeksowane.

Te właściwości należy pozostawić przy wartościach domyślnych 0, chyba że użytkownik zostanie poproszony o zmianę ich przez Centrum wsparcia produktu IBM .

Należy zauważyć, że produkt WebSphere Application Server ma jedną dodatkową właściwość zaawansowanej puli połączeń o nazwie `Number of shared partitions`. Ta właściwość określa liczbę partycji używanych do przechowywania współużytkowanych połączeń. Jednak ponieważ połączenia JMS są zawsze niewspółużytkowane, ta właściwość nie ma zastosowania.

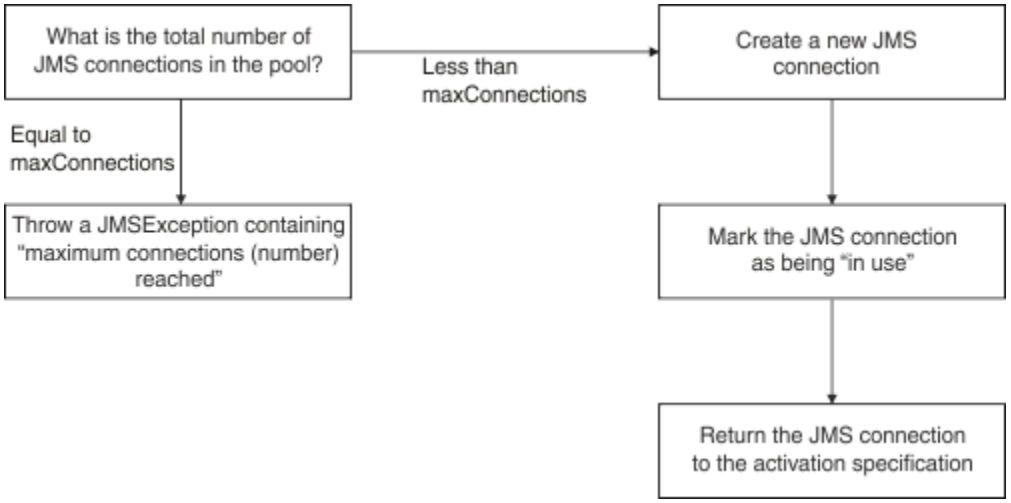
Przykłady korzystania z puli połączeń
Komponent portu nasłuchiwania komponentu bean sterowanego komunikatami oraz aplikacje, które wykonują komunikaty wychodzące, korzystają z puli połączeń produktu JMS .

Rysunek 49 na stronie 305 przedstawia sposób działania puli połączeń dla produktów WebSphere Application Server V7.5 i V8.0.



Rysunek 49. WebSphere Application Server V7.5 i V8.0 -sposób działania puli połączeń

Rysunek 50 na stronie 305 przedstawia sposób działania puli połączeń dla produktu WebSphere Application Server V8.5.



Rysunek 50. WebSphere Application Server V8.5 -sposób działania puli połączeń

W jaki sposób porty nasłuchiwania MDB używają puli połączeń

Załóżmy, że istnieje komponent MDB wdrożony w systemie WebSphere Application Server Network Deployment, który używa IBM MQ jako dostawcy JMS . Komponent MDB jest wdrażany dla portu nasłuchiwania, który korzysta z fabryki połączeń wywołanej na przykład `jms/CF1`, która ma właściwość *maksymalna liczba połączeń* ustawioną na wartość 2, co oznacza, że tylko dwa połączenia mogą być tworzone z tej fabryki w dowolnym momencie.

Po uruchomieniu portu nasłuchiwania port podejmuje próbę utworzenia połączenia z produktem IBM MQprzy użyciu fabryki połączeń produktu`jms/CF1` .

Aby to zrobić, port żąda połączenia od menedżera połączeń. Ponieważ jest to pierwsza używana fabryka połączeń produktu `jms/CF1` , nie ma żadnych połączeń w puli połączeń z produktem `jms/CF1` , dlatego

menedżer połączeń tworzy nowy, na przykład c1. Należy zauważyć, że to połączenie istnieje przez cały czas życia portu nasłuchiwania.

Należy rozważyć sytuację, w której port nasłuchiwania jest zatrzymany przy użyciu Konsoli administracyjnej serwera WebSphere Application Server . W takim przypadku menedżer połączeń nawiąże połączenie i umieszcza go z powrotem w puli. Jednak połączenie z produktem IBM MQ pozostaje otwarte.

Po zrestartowaniu portu nasłuchiwania port po raz kolejny zwraca się do menedżera połączeń o połączenie z menedżerem kolejek. Ponieważ w tej chwili istnieje połączenie (c1) w puli wolnych pul, menedżer połączeń pobiera to połączenie z puli i udostępnia go do portu nasłuchiwania.

Załóżmy teraz, że na serwerze aplikacji wdrożono drugi komponent MDB, który korzysta z innego portu nasłuchiwania.

Załóżmy, że użytkownik próbuje uruchomić trzeci port nasłuchiwania, który jest również skonfigurowany do korzystania z fabryki połączeń produktu jms/CF1 . Trzeci port nasłuchiwania żąda połączenia od menedżera połączeń, który wygląda w puli wolnej dla jms/CF1 i znajduje, że jest pusty. 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 przez 180 sekund (wartość domyślna właściwości limitu czasu połączenia), aby połączenie stało się dostępne.

Jeśli jednak zostanie zatrzymany pierwszy port nasłuchiwania, jego połączenie c1 zostanie umieszczone w wolnej puli dla produktu jms/CF1. Menedżer połączeń pobiera to połączenie i przekazuje je do trzeciego programu nasłuchującego.

Jeśli teraz zostanie uruchomiony restart pierwszego obiektu nasłuchiwania, ten program nasłuchujący musi czekać na zatrzymanie jednego z pozostałych portów nasłuchiwania przed zrestartowaniem pierwszego programu nasłuchującego. Jeśli żaden z działających portów nasłuchiwania nie zostanie zatrzymany w ciągu 180 sekund, pierwszy program nasłuchujący otrzyma błąd ConnectionWaitTimeoutException i zostanie zatrzymany.

W jaki sposób aplikacje, które wykonują komunikaty wychodzące, korzystają z puli połączeń

W przypadku tej opcji należy założyć, że istnieje pojedynczy komponent EJB, na przykład EJB1, zainstalowany na serwerze aplikacji. Komponent bean implementuje metodę o nazwie sendMessage() , wykonując następujące czynności:

- Tworzenie połączenia JMS z produktem IBM MQ z fabryki jms/CF1przy użyciu produktu connectionFactory.createConnection().
- Tworzenie sesji JMS z połączenia.
- Tworzenie producenta komunikatów z sesji.
- Wysyłanie wiadomości.
- Zamykanie producenta.
- Zamykanie sesji.
- Zamknięcie połączenia, wywołując program connection.close().

Załóżmy, że wolna pula dla fabryki jms/CF1 jest pusta. Gdy komponent EJB zostanie wywołany po raz pierwszy, komponent bean próbuje utworzyć połączenie z produktem IBM MQ z fabryki jms/CF1. Ponieważ pula wolnych pul dla fabryki jest pusta, menedżer połączeń tworzy nowe połączenie i nadaje mu wartość EJB1.

Tuż przed zakończeniem metody, metoda wywołuje metodę connection.close(). Zamiast zamykanego c1menedżer połączeń pobiera połączenie i umieszcza go w puli wolnej dla produktu jms/CF1.

Następnym razem, gdy zostanie wywołana sendMessage() , metoda connectionFactory.createConnection() zwraca c1 do aplikacji.

Załóżmy, że istnieje druga instancja komponentu EJB działającego w tym samym czasie, co pierwsza instancja. Gdy obie instancje wywołują produkt `sendMessage()`, dwa połączenia są tworzone z fabryki połączeń produktu `jms/CF1` .

Załóżmy teraz, że tworzona jest trzecia instancja komponentu bean. Gdy trzeci komponent bean wywołuje produkt `sendMessage()`, metoda wywołuje metodę `connectionFactory.createConnection()` , aby utworzyć połączenie z produktu `jms/CF1`.

Istnieją jednak obecnie dwa połączenia utworzone z produktu `jms/CF1`, które są równe wartości maksymalnej liczby połączeń dla tej fabryki. Oznacza to, że metoda `createConnection()` czeka przez 180 sekund (wartość domyślna właściwości limitu czasu połączenia), aby połączenie stało się dostępne.

Jeśli jednak metoda `sendMessage()` dla pierwszego wywołania komponentu EJB `connection.close()` i kończy działanie, to połączenie, które było używane, `c1`zostanie ponownie umieszczone w puli wolnych połączeń. Menedżer połączeń pobiera z wolnej puli połączenie z powrotem i przekazuje je do trzeciego komponentu EJB. Wywołanie z tego komponentu bean do programu `connectionFactory.createConnection()` następnie zwraca, pozwalając na zakończenie metody `sendMessage()`.

Porty nasłuchiwania MDB i komponenty EJB korzystające z tej samej puli połączeń

W dwóch poprzednich przykładach przedstawiono sposób, w jaki porty nasłuchiwania i komponenty EJB mogą korzystać z puli połączeń w izolacji. Można jednak mieć zarówno port nasłuchiwania, jak i komponent EJB działający w obrębie tego samego serwera aplikacji i tworzący połączenia produktu JMS przy użyciu tej samej fabryki połączeń.

Należy wziąć pod uwagę konsekwencje tej sytuacji

Kluczową rzeczą jest to, aby fabryka połączeń była współużytkowana przez port nasłuchiwania i komponent EJB.

Załóżmy na przykład, że w tym samym czasie jest uruchomiony program nasłuchujący i komponent EJB. Oba te elementy korzystają z fabryki połączeń produktu `jms/CF1` , co oznacza, że osiągnięto limit połączeń określony przez właściwość maksymalnej liczby połączeń dla tej fabryki.

Jeśli użytkownik podejmie próbę uruchomienia innego portu nasłuchiwania lub innej instancji komponentu EJB, musi czekać na powrót połączenia do puli wolnych połączeń dla produktu `jms/CF1`.

Wątki konserwacji puli połączeń
Powiązana z każdą wolną pulą połączeń jest wątek konserwacji puli, który monitoruje pulę wolnych pul, aby zapewnić, że połączenia w nim są nadal poprawne.

Jeśli wątek konserwacji puli podejmie decyzję o tym, że połączenie w puli musi zostać usunięte, wątek fizycznie zamknie połączenie JMS z IBM MQ.

Jak działa wątek konserwacji puli

Zachowanie wątku konserwacji puli jest określane przez wartość czterech właściwości puli połączeń:

Limit wieku
Czas, przez jaki połączenie pozostaje otwarte.

Minimalna liczba połączeń
Minimalna liczba połączeń, które menedżer połączeń utrzymuje w wolnej puli w fabryce połączeń.

Czas między sprawdzeniami
Jak często działa wątek konserwacji puli.

Czas niewykorzystany
Jak długo połączenie pozostaje w wolnej puli, zanim zostanie zamknięte.

Domyślnie wątek utrzymywany w puli jest uruchamiany co 180 sekund, chociaż wartość ta może zostać zmieniona przez ustawienie właściwości puli połączeń **`Reap time`** .

Wątek konserwacji patrzy na każde połączenie w puli, sprawdza, jak długo znajduje się w puli, oraz ile czasu upłynęło od momentu jego utworzenia i ostatniego użycia.

Jeśli połączenie nie zostało użyte przez okres dłuższy niż wartość właściwości **Unused timeout** dla puli połączeń, wątek konserwacji sprawdza liczbę połączeń znajdujących się obecnie w puli wolnych połączeń. Jeśli ten numer jest następujący:

- Wartość większa niż wartość **Minimum connections**powoduje zamknięcie połączenia przez menedżera połączeń.
- Wartość równa **Minimum connections**oznacza, że połączenie nie jest zamknięte i pozostaje w puli wolnych.

Wartością domyślną właściwości **Minimum connections** jest *1*, co oznacza, że ze względu na wydajność menedżer połączeń zawsze próbuje zachować co najmniej jedno połączenie w wolnej puli.

Właściwość **Unused timeout** ma wartość domyślną 1800 sekund. Domyślnie, jeśli połączenie jest wstawiane do puli wolnych i nie jest używane ponownie przez co najmniej 1800 sekund, to połączenie jest zamykane, pod warunkiem, że zamykane jest, pozostawia co najmniej jedno połączenie w wolnej puli.

Ta procedura zapobiega stałowaniu się nieużywanych połączeń. Aby wyłączyć tę funkcję, należy ustawić właściwość **Unused timeout** na wartość zero.

Jeśli połączenie znajduje się w puli wolnych pul, a czas, który upłynął od momentu jego utworzenia, jest większy niż wartość właściwości **Aged timeout** dla puli połączeń, to jest on zamykany niezależnie od tego, jak długo został on użyty od ostatniego użycia.

Domyślnie właściwość **Aged timeout** jest ustawiona na zero, co oznacza, że wątek konserwacji nigdy nie wykonuje tej operacji sprawdzania. Połączenia, które były w pobliżu dłużej niż w przypadku właściwości **Aged timeout** , są usuwane niezależnie od tego, 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 widać, że wątek konserwacji puli zajmuje dużo pracy, gdy jest aktywny, szczególnie w przypadku dużej liczby połączeń w wolnej puli fabryki połączeń.

Załóżmy na przykład, że istnieją trzy fabryki połączeń produktu JMS , z właściwością **Maximum connections** ustawioną na wartość 10 dla każdej fabryki. Co 180 sekund, trzy wątki konserwacji puli stają się aktywne i skanują pule wolnych pul odpowiednio dla każdej fabryki połączeń. Jeśli wolne pule mają wiele połączeń, wątki konserwacji mają dużo pracy do wykonania, co może znacznie wpłynąć na wydajność.

Można wyłączyć wątek konserwacji puli dla pojedynczej wolnej puli połączeń, ustawiając jej właściwość **Reap time** na wartość zero.

Wyłączenie wątku konserwacji oznacza, że połączenia nigdy nie są zamykane, nawet jeśli upłynął czas **Unused timeout** . Jednak połączenia mogą być nadal zamykane, jeśli program **Aged timeout** został przekazany.

Gdy aplikacja zakończy połączenie, menedżer połączeń sprawdza, jak długo istnieje połączenie, a jeśli ten okres jest dłuższy niż wartość właściwości **Aged timeout** , menedżer połączeń zamknie połączenie, a nie zwróci go do wolnej puli.

Implikacje transakcyjne limitu czasu Aged

Zgodnie z opisem w poprzedniej sekcji właściwość **Aged timeout** określa, jak długo połączenie z dostawcą JMS pozostaje otwarte, zanim menedżer połączeń zamknie go.

Wartością domyślną dla właściwości **Aged timeout** jest zero, co oznacza, że połączenie nigdy nie zostanie zamknięte, ponieważ jest zbyt stare. Wartość właściwości **Aged timeout** należy pozostawić na tej wartości, ponieważ włączenie produktu **Aged timeout** może mieć wpływ na transakcje podczas korzystania z produktu JMS wewnątrz komponentów EJB.

W programie JMSjednostką transakcji jest JMS *sesja*, która jest tworzona na podstawie *połączenia* JMS . Jest to JMS *sesja* , która jest ujęta w transakcje, a nie *połączenie* JMS .

Ze względu na konstrukcję serwera aplikacji połączenia produktu JMS mogą być zamykane, ponieważ serwer **Aged timeout** upłynął, nawet jeśli sesje produktu JMS utworzone na podstawie tego połączenia są zaangażowane w transakcję.

Zamknięcie połączenia JMS powoduje wycofanie wszystkich zaległych prac transakcyjnych w sesjach JMS, zgodnie z opisem w specyfikacji JMS . Jednak serwer aplikacji nie jest świadomy, że sesje produktu JMS utworzone na podstawie połączenia nie są już poprawne. Gdy serwer próbuje użyć sesji w celu zatwierdzenia lub wycofania transakcji, występuje `IllegalStateException` .

Ważne: Jeśli produkt **Aged timeout** ma być używany z połączeniami produktu JMS z poziomu komponentów EJB, należy upewnić się, że wszystkie prace JMS są jawnie zatwierdzone w sesji produktu JMS przed użyciem metody EJB, która wykonuje operacje wyjścia z operacji JMS .

Przykłady wątków konserwacji puli
Korzystając z przykładu komponentów EJB (Enterprise Java Bean), można zrozumieć, w jaki sposób działa wątek konserwacji puli. Należy pamiętać, że można również używać komponentów bean sterowanych komunikatami (Message Driven Beans-MDBs) i portów nasłuchiwania, ponieważ wszystko, czego potrzebujesz, to sposób na uzyskanie połączeń w wolnej puli.

Więcej informacji na temat metody `sendMessage()` zawiera sekcja "W jaki sposób aplikacje, które wykonują komunikaty wychodzące, korzystają z puli połączeń" na stronie 306 .

Fabryka połączeń została skonfigurowana przy użyciu następujących wartości:

- **Reap time** przy domyślnej wartości 180 sekund
- **Aged timeout** w domyślnej wartości zero sekund
- **Unused timeout** ustawione na 300 sekund

Po uruchomieniu serwera aplikacji zostanie wywołana metoda `sendMessage()` .

Metoda tworzy połączenie o nazwie, na przykład `c1`, przy użyciu fabryki `jms/CF1`, używa tej fabryki do wysłania komunikatu, a następnie wywołuje `connection.close()`, co powoduje, że produkt `c1` zostanie umieszczony w wolnej puli.

Po upływie 180 sekund wątek konserwacji puli zostanie uruchomiony i będzie wyglądał na pulę wolnych połączeń serwera `jms/CF1` . W puli znajduje się wolne połączenie `c1` , więc wątek konserwacji wygląda w momencie, gdy połączenie zostało wstawione, i porównuje je z bieżącą godziną.

Miną 180 sekund od momentu umieszczenia połączenia w puli wolnej, która jest mniejsza od wartości właściwości **Unused timeout** dla `jms/CF1`. W związku z tym wątek konserwacji pozostawia połączenie w pojedynkę.

180 sekund później wątek konserwacji puli jest uruchamiany ponownie. Wątek konserwacji znajduje połączenie `c1`i określa, że połączenie było w puli przez 360 sekund, co jest dłuższe niż zestaw wartości **Unused timeout** , więc menedżer połączeń zamknie połączenie.

Jeśli metoda `sendMessage()` zostanie teraz uruchomiona ponownie, gdy aplikacja wywołuje produkt `connectionFactory.createConnection()` , menedżer połączeń utworzy nowe połączenie z produktem IBM MQ , ponieważ pula połączeń wolnych 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** jest ustawiona na zero.

W jaki sposób działa właściwość **Aged timeout** ?

W poniższym przykładzie założono, że ustawiono następujące wartości:

- Właściwość **Aged timeout** do 300 sekund
- Właściwość **Unused timeout** do zera.

Metoda `sendMessage()` jest wywoływana, a ta metoda próbuje utworzyć połączenie z fabryki połączeń produktu `jms/CF1` .

Ponieważ pula wolnych pul dla tej fabryki jest pusta, menedżer połączeń tworzy nowe połączenie `c1`i zwraca je do aplikacji. When `sendMessage()` calls `connection.close()`, `c1` is put back into the free connection pool.

180 sekund później, wątek konserwacji puli działa. Wątek znajduje `c1` w puli połączeń wolnych, a następnie sprawdza, jak dawno temu utworzono. Połączenie istnieje przez 180 sekund, co jest mniejsze niż **Aged timeout**, więc wątek konserwacji puli pozostawia go w spokoju i wraca do stanu uśpienia.

60 sekund później, `sendMessage()` jest ponownie wywoływana. Tym razem, gdy metoda wywołuje program `connectionFactory.createConnection()`, menedżer połączeń wykrywa, że istnieje połączenie `c1`dostępne w puli wolnych pul dla produktu `jms/CF1`. Menedżer połączeń pobiera `c1` z wolnej puli i umożliwia nawiązanie połączenia z aplikacją.

Połączenie jest zwracane do puli wolnych, gdy program `sendMessage()` kończy działanie. 120 sekund później wątek konserwacji puli ponownie się obudzi, skanuje zawartość wolnej puli dla `jms/CF1` i wykrywa `c1`.

Mimo że połączenie zostało użyte tylko 120 sekund temu, wątek konserwacji puli zamknie połączenie, ponieważ połączenie istnieje przez łącznie 360 sekund, co jest dłuższe niż wartość 300 drugiej wartości ustawionej dla właściwości **Aged timeout** .

Sposób, w jaki właściwość Minimalna liczba połączeń wpływa na wątek konserwacji puli

Korzystając z przykładu “W jaki sposób porty nasłuchiwania MDB używają puli połączeń” na stronie 305 , założono, że na serwerze aplikacji wdrożono dwa komponenty MDB, a każdy z nich korzysta z innego portu nasłuchiwania.

Każdy port nasłuchiwania jest skonfigurowany pod kątem korzystania z fabryki połączeń produktu `jms/CF1` , którą skonfigurowano przy użyciu następujących portów:

- Właściwość **Unused timeout** ustawiona na 120 sekund
- Właściwość **Reap time** ustawiona na 180 sekund
- Właściwość **Minimum connections** ustawiona na wartość 1

Załóżmy, że pierwszy program nasłuchujący został zatrzymany, a jego połączenie `c1` jest umieszczane w wolnej puli. 180 sekund później wątek konserwacji puli się obudzi, skanuje zawartość wolnej puli dla produktu `jms/CF1`i wykrywa, że produkt `c1` jest w puli wolnych od wartości większej niż wartość właściwości **Unused timeout** dla fabryki połączeń.

Jednak przed zamknięciem `c1`wątek konserwacji puli sprawdza, ile połączeń pozostanie w puli, jeśli połączenie zostanie odrzucone. Ponieważ produkt `c1` jest jedynym połączeniem w puli wolnych połączeń, menedżer połączeń nie jest zamykany, ponieważ w ten sposób liczba połączeń, które pozostają w puli wolnych połączeń, jest mniejsza niż wartość ustawiona dla **Minimum connections**.

Załóżmy teraz, że drugi obiekt nasłuchiwania został zatrzymany. Wolna pula połączeń zawiera teraz dwa wolne połączenia- `c1` i `c2`.

180 sekund później wątek konserwacji puli jest uruchamiany ponownie. W tym czasie program `c1` jest w puli wolnych połączeń przez 360 sekund, a serwer `c2` przez 180 sekund.

Wątek konserwacji puli sprawdza program `c1` i wykrywa, że znajduje się on w puli dłużej niż wartość właściwości **Unused timeout** .

Następnie wątek sprawdza, ile połączeń znajduje się w puli wolnych pul, i porównuje je z wartością właściwości **Minimum connections** . Ponieważ pula zawiera dwa połączenia, a wartość **Minimum connections** jest ustawiona na 1, menedżer połączeń zamknie produkt `c1`.

Wątek konserwacji będzie teraz wyglądał na `c2`. Ta pula połączeń jest również dostępna w puli połączeń bezpłatnych dłużej niż wartość właściwości **Unused timeout** . Jednak ponieważ zamknięcie systemu

c2 pozostawi wolną pulę połączeń mniejszą od ustawionej w niej minimalnej liczby połączeń, menedżer połączeń pozostawi sam produkt c2 .

Połączenia JMS i IBM MQ
Informacje na temat używania produktu IBM MQ jako dostawcy JMS .

Korzystanie z transportu powiązań

Jeśli fabryka połączeń została skonfigurowana tak, aby korzystała z transportu powiązań, każde połączenie JMS nawiązuje konwersację (zwaną również **hconn**) z produktem IBM MQ. W konwersacji używana jest komunikacja międzyprocesowa (lub pamięć współużytkowana) do komunikacji z menedżerem kolejek.

Korzystanie z transportu klienta

Gdy fabryka połączeń dostawcy przesyłania komunikatów produktu IBM MQ została skonfigurowana pod kątem korzystania z 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 przy użyciu trybu normalnego dostawcy przesyłania komunikatów produktu IBM MQ , możliwe jest utworzenie wielu połączeń produktu JMS z fabryki połączeń w celu współużytkowania połączenia TCP/IP z produktem IBM MQ. Więcej informacji na ten temat zawiera sekcja “Współużytkowanie połączenia TCP/IP w produkcie IBM MQ classes for JMS” na stronie 313.

Aby określić maksymalną liczbę kanałów klienta używanych przez połączenia JMS w dowolnym momencie, 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 łączenia się z tym samym menedżerem kolejek produktu IBM MQ przy użyciu tego samego kanału IBM MQ .

Te fabryki korzystają z domyślnych właściwości puli połączeń, co oznacza, że wartość *Maksymalna liczba połączeń* jest ustawiona na 10. Jeśli wszystkie połączenia są używane zarówno z jms/CF1 , jak i jms/CF2 w tym samym czasie, między serwerem aplikacji a produktem IBM MQzostanie nawiązane 20 konwersacji.

Jeśli fabryka połączeń łączy się z menedżerem kolejek przy użyciu trybu normalnego dostawcy przesyłania komunikatów produktu IBM MQ , to maksymalna liczba połączeń TCP/IP, które mogą istnieć 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 , to maksymalna liczba połączeń TCP/IP między serwerem aplikacji a IBM MQ dla tych fabryk połączeń będzie wynosić 20 (jeden dla każdego połączenia JMS w pulach połączeń dla dwóch fabryk).

Pojęcia pokrewne
“użycieIBM MQ classes for JMS” na stronie 83
IBM MQ classes for Java Message Service (IBM MQ classes for JMS) jest dostawcą JMS dostarczonym z produktem IBM MQ. Oprócz implementowania interfejsów zdefiniowanych w pakiecie javax.jms produkt IBM MQ classes for JMS udostępnia dwa zestawy rozszerzeń do interfejsu API produktu JMS .

Zestawianie obiektów w środowisku Java SE
W przypadku produktu Java SE (lub z innymi ramami, takimi jak Spring) modele programistyczne są niezwykle elastyczne. Dlatego też jedna strategia łączenia nie pasuje do wszystkich. Należy rozważyć, czy istnieje struktura, która mogłaby zrobić dowolną formę łączenia, na przykład Spring.

W przeciwnym razie może to zająć logika aplikacji. Zadaj sobie pytanie, jak złożony jest sam wniosek? Najlepiej jest zrozumieć aplikację i to, co wymaga od połączenia z systemem przesyłania komunikatów. Aplikacje często są również zapisywane w obrębie własnego kodu opakowania wokół podstawowego interfejsu API produktu JMS .

Choć może to być bardzo rozsądne podejście, i może ukrywać złożoność, warto pamiętać, że może ona wprowadzać problemy. Na przykład ogólna metoda `getMessage()` , która jest często wywoływana, nie powinna tylko otwierać i zamykać konsumentów.

Punkty, które należy wziąć pod uwagę:

- Jak długo aplikacja będzie potrzebować dostępu do produktu IBM MQ? Przez cały czas, albo tylko od czasu do czasu.
- Jak często komunikaty będą wysyłane? Im rzadziej, tym bardziej pojedyncze połączenie z produktem IBM MQ może być współużytkowane.
- Zerwany wyjątek jest zwykle oznaką potrzebną do ponownego utworzenia połączenia z puli. Co z:
  - Wyjątki zabezpieczeń lub host nie jest dostępny
  - Wyjątki pełnej kolejki
- Jeśli wystąpi wyjątek zerwany z połączeniem, co powinno się stać z innymi wolnymi połączeniami w puli? Czy powinny one zostać zamknięte i ponownie utworzone?
- Jeśli używany jest protokół TLS, na przykład: jak długo ma pozostać otwarte pojedyncze połączenie?
- W jaki sposób połączenie zbiorcze będzie identyfikować się w taki sposób, że administrator menedżera kolejek może nawiązać połączenie i śledzić je z powrotem.

Należy wziąć pod uwagę wszystkie obiekty produktu JMS do zestawiania oraz pulę obiektów, gdy tylko możliwe jest to działanie. Obiekty te obejmują:

- Połączenia serwera JMS
- Sesja
- Konteksty
- Producenci i konsumenci wszystkich różnych typów

Jeśli używany jest transport klienta, połączenia, sesje i konteksty produktu JMS będą używały gniazd podczas komunikacji z menedżerem kolejek produktu IBM MQ . Dzięki zestawianiu tych obiektów oszczędności są związane z liczbą przychodzących połączeń IBM MQ (`hConns`) z menedżerem kolejek i zmniejszeniem liczby instancji kanału.

Użycie transportu powiązań do menedżera kolejek powoduje całkowite usunięcie warstwy sieciowej. Jednak wiele aplikacji korzysta z transportu klienta w celu zapewnienia bardziej wysokiej dostępności, a obciążenie jest zrównoważone, a konfiguracja jest zrównoważona.

Producenci JMS i konsumenci otwierają miejsca docelowe w menedżerze kolejek. Jeśli otwartych jest mniejsza liczba kolejek lub tematów, a wiele części aplikacji korzysta z tych obiektów, łączenie tych obiektów może być użyteczne.

Z perspektywy IBM MQ proces ten zapisuje sekwencję operacji MQOPEN i MQCLOSE.

Połączenia, sesje i konteksty

Te obiekty obsługują wszystkie uchwyty połączeń produktu IBM MQ z menedżerem kolejek i są generowane z poziomu `ConnectionFactory`. Istnieje możliwość dodania logiki do aplikacji w celu ograniczenia liczby połączeń, a także innych obiektów utworzonych z pojedynczej fabryki połączeń do określonej liczby.

W aplikacji można użyć prostej struktury danych, która będzie zawierać utworzone połączenia. Kod aplikacji, który musi użyć jednej z tych struktur danych, może *wymeldowanie* obiekt, który ma być używany.

Weź pod uwagę następujące czynniki:

- Kiedy połączenia powinny zostać usunięte z puli? Generalnie należy utworzyć obiekt nasłuchiwania wyjątków dla połączenia. Gdy ten proces nasłuchiwania jest wywoływany w celu przetworzenia wyjątku, należy ponownie utworzyć połączenie oraz wszystkie sesje utworzone na podstawie tego połączenia.
- Jeśli pakiet CCDT jest używany do równoważenia obciążenia, połączenia mogą być używane do różnych menedżerów kolejek. Może to mieć zastosowanie w przypadku wymagań dotyczących zestawiania połączeń.

Należy pamiętać, że specyfikacja JMS wskazuje, że jest to błąd programistyczny dla wielu wątków w celu uzyskania 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 jest używany tylko przez jeden wątek naraz.

Producenci i konsumenci

Każdy tworzony producent i konsument otwiera miejsce docelowe w menedżerze kolejek. Jeśli to samo miejsce docelowe zostanie użyte dla różnych zadań, ma to sens, aby zachować otwarte obiekty konsumenta lub producenta. Należy zamknąć obiekt tylko wtedy, gdy cała praca jest wykonywana.

Chociaż otwarcie i zamknięcie miejsca docelowego to krótkie operacje, to jeśli są one wykonywane często, to czas ten może zostać dodany.

Zasięg tych obiektów znajduje się w obrębie sesji lub kontekstu, z których są tworzone, dlatego muszą być one przechowywane w tym zasięgu. Ogólnie, wnioski są pisane tak, że jest to całkiem proste do zrobienia.

Monitorowanie

W jaki sposób aplikacje będą monitorować ich pule obiektów? Odpowiedź na to pytanie jest w dużej mierze zdeterminowana przez złożoność rozwiązania w zakresie łączenia wdrożonego.

Jeśli rozważana jest implementacja zestawiania serwerów JavaEE, istnieje duża liczba opcji, w tym:

- Bieżąca wielkość pul
- Obiekty czasu spędzone w nich
- Czyszczenie basenów
- Odświeżanie połączeń

Należy również rozważyć, w jaki sposób w menedżerze kolejek zostanie wyświetlona pojedyncza sesja ponownie używana. Istnieją właściwości fabryki połączeń służące do identyfikowania aplikacji (takiej jak `appName`), która może być przydatna.

“użycieIBM MQ classes for JMS” na stronie 83
IBM MQ classes for Java Message Service (IBM MQ classes for JMS) jest dostawcą JMS dostarczonym z produktem IBM MQ. Oprócz implementowania interfejsów zdefiniowanych w pakiecie javax.jms produkt IBM MQ classes for JMS udostępnia dwa zestawy rozszerzeń do interfejsu API produktu JMS .

Współużytkowanie połączenia TCP/IP w produkcie IBM MQ classes for JMS
W celu współużytkowania pojedynczego połączenia TCP/IP można utworzyć wiele instancji kanału MQI.

Aplikacje działające w tym samym środowisku wykonawczym produktu Java i korzystające z adaptera zasobów produktu IBM MQ classes for JMS lub IBM MQ w celu nawiązania połączenia z menedżerem kolejek przy użyciu transportu CLIENT mogą być udostępniane do współużytkowania tej samej instancji kanału.

Między instancjami kanału a połączeniami TCP/IP istnieje relacja jeden do jednego. Dla każdej instancji kanału tworzone jest jedno połączenie TCP/IP.

Jeśli kanał jest zdefiniowany z parametrem **SHARECNV** ustawionym na wartość większą niż 1, to ta liczba konwersacji może współużytkować instancję kanału. Aby włączyć fabrykę połączeń lub specyfikację aktywowania w celu użycia tej funkcji, należy ustawić właściwość **SHARECONVALLOWED** na wartość YES.

Każde połączenie JMS i sesja JMS utworzone przez aplikację JMS tworzy własną konwersację z menedżerem kolejek.

Po uruchomieniu specyfikacji aktywowania adapter zasobów produktu IBM MQ rozpoczyna konwersację z menedżerem kolejek w celu użycia specyfikacji aktywowania. Każda sesja serwera w puli sesji serwera, która jest powiązana ze specyfikacją aktywowania, rozpoczyna również konwersację z menedżerem kolejek.

Atrybut SHARECNV jest najlepszym sposobem podejścia do współużytkowania połączeń. W związku z tym, gdy wartość SHARECNV większa niż 0 jest używana z produktem IBM MQ classes for JMS, nie ma gwarancji, że nowe żądanie połączenia zawsze będzie współużytkował nawiązane już połączenie.

Obliczanie liczby instancji kanału

Aby określić maksymalną liczbę instancji kanałów utworzonych przez aplikację, należy użyć następujących formuł:

Specyfikacje aktywowania

Liczba instancji kanału = (*maxPoolDepth_value* + 1)/ *SHARECNV_value*

Gdzie *maxPoolwartość_zależna* to wartość właściwości **maxPoolDepth** , a *SHARECNV_value* to wartość właściwości **SHARECNV** w kanale, która jest używana przez specyfikację aktywowania.

Inne aplikacje produktu JMS

Liczba instancji kanału = (*jms_connections* + *jms_sessions*)/ *SHARECNV_value*

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, która jest używana przez specyfikację aktywowania.

Przykłady

W poniższych przykładach przedstawiono sposób użycia formuł w celu obliczenia liczby instancji kanału, które są tworzone w menedżerze kolejek przez aplikacje przy użyciu adaptera zasobów IBM MQ classes for JMS lub IBM MQ .

JMS przykład zastosowania

Połączenie aplikacji JMS łączy się z menedżerem kolejek za pomocą transportu CLIENT i tworzy połączenie JMS i trzy sesje produktu JMS . Kanał, za pomocą którego aplikacja używa do połączenia z menedżerem kolejek, ma właściwość **SHARECNV** ustawioną na wartość 10. Gdy aplikacja jest uruchomiona, między aplikacją a menedżerem kolejek i jedną instancją kanału są dostępne cztery konwersacje. Wszystkie cztery konwersacje współużytkują instancję kanału.

V 9.1.3 W produkcie IBM MQ 9.1.3, jeśli aplikacje są skonfigurowane jako `reconnectable`, instancje kanałów mogą być współużytkowane tylko między pokrewnymi obiektami produktu JMS , tj. połączeniem JMS i powiązanymi z nim sesjami JMS . Może to wymagać skonfigurowania dodatkowych instancji kanałów w celu obsługi takich aplikacji.

Jeśli na przykład aplikacja korzysta z pojedynczego połączenia JMS i pojedynczej sesji JMS , a używany kanał ma wartość **SHARECNV** równą 10, przed IBM MQ 9.1.3, do pięciu instancji aplikacji może być współużytkowana pojedyncza instancja kanału. Jest to nadal w przypadku produktu IBM MQ 9.1.3 lub nowszej wersji, jeśli aplikacja nie jest skonfigurowana jako `reconnectable`, ale jeśli aplikacja jest skonfigurowana jako `reconnectable`, każda instancja aplikacji będzie wymagała własnej instancji kanału, w związku z czym konieczne będzie pięć instancji kanału łącznie.

Przykład specyfikacji aktywowania

Specyfikacja aktywowania łączy się z menedżerem kolejek przy użyciu transportu CLIENT. Specyfikacja aktywowania jest skonfigurowana z właściwością **maxPoolDepth** ustawioną na wartość 10. Kanał, którego konfiguracja aktywowania jest skonfigurowana do użycia, 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, a jedna dla specyfikacji aktywowania). Liczba instancji kanału, które są używane przez specyfikację aktywowania to 2.

Przykład specyfikacji aktywowania

Specyfikacja aktywowania łączy się z menedżerem kolejek przy użyciu transportu CLIENT. Specyfikacja aktywowania jest skonfigurowana z właściwością **maxPoolDepth** ustawioną na wartość 5. Kanał, który jest skonfigurowany do używania specyfikacji aktywowania, ma właściwość **SHARECNV** ustawioną na 0. Gdy specyfikacja aktywowania jest uruchomiona i jednocześnie przetwarza 5 komunikatów, liczba konwersacji między specyfikacją aktywowania a menedżerem kolejek wynosi 6 (pięć konwersacji dla sesji serwera, a jedna dla specyfikacji aktywowania). Liczba instancji kanału, które są używane przez specyfikację aktywowania to 6, ponieważ właściwość **SHARECNV** w kanale jest ustawiona na 0, każda konwersacja korzysta z własnej instancji kanału.

Zadania pokrewne

“Określanie liczby połączeń TCP/IP, które są tworzone z WebSphere Application Server do IBM MQ” na stronie 512

Za pomocą funkcji współużytkowania konwersacji wiele konwersacji może współużytkować instancje kanału MQI, co jest również znane jako połączenie 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 się wiązać.

Gdy aplikacja IBM MQ classes for JMS próbuje nawiązać połączenie z menedżerem kolejek produktu IBM MQ w trybie klienta, firewall może zezwalać tylko na połączenia, które pochodzą z określonych portów lub z zakresu portów. W takiej sytuacji można użyć właściwości LOCALADDRESS obiektu fabryki połączeń ConnectionFactory, QueueConnectionlub obiektu fabryki TopicConnectionw celu określenia portu lub zakresu portów, z którymi aplikacja może się wiązać.

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 nawiąże połączenie z menedżerem kolejek, aplikacja wiąże 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 właściwości LOCALADDRESS, aby określić, który interfejs sieciowy musi być używany dla połączenia.

W przypadku połączenia w czasie rzeczywistym z brokerem właściwość LOCALADDRESS ma znaczenie tylko wtedy, gdy używana jest funkcja rozsyłania grupowego. W takim przypadku można użyć tej właściwości w celu określenia, który lokalny interfejs sieciowy musi być używany dla połączenia, ale wartość właściwości nie może zawierać numeru portu ani zakresu numerów portów.

W przypadku ograniczenia zakresu portów mogą wystąpić błędy połączenia. Jeśli wystąpi błąd, zgłaszany jest wyjątek JMSException z osadzonym wyjątkiem MQException, który zawiera kod przyczyny produktu IBM MQ MQRC_Q_MGR_NOT_AVAILABLE i następujący komunikat:

```
Próba nawiązania połączenia przez gniazdo nie powiodła się ze względu na ograniczenia właściwości
LOCAL_ADDRESS_PROPERTY
```

Błąd może wystąpić, jeśli używane są wszystkie porty w podanym zakresie lub jeśli podany adres IP, nazwa hosta lub numer portu nie są poprawne (na przykład jest to ujemny numer portu).

Ponieważ program IBM MQ classes for JMS może tworzyć połączenia inne niż te wymagane przez aplikację, należy zawsze rozważyć określenie zakresu portów. Na ogół każda sesja utworzona przez aplikację wymaga jednego portu, a IBM MQ classes for JMS może wymagać trzech lub czterech dodatkowych portów. Jeśli wystąpi błąd połączenia, należy zwiększyć zakres portów.

Zestawianie połączeń, które jest używane domyślnie w produkcie IBM MQ classes for JMS, może mieć wpływ na szybkość, z jaką porty mogą być ponownie wykorzystywane. W związku z tym może wystąpić błąd połączenia, gdy porty są zwalniane.

Kompresja kanału w produkcie IBM MQ classes for JMS
Aplikacja IBM MQ classes for JMS może używać narzędzi IBM MQ do kompresowania nagłówka komunikatu lub danych.

Kompresowanie danych, które przepływają na kanale IBM MQ , może poprawić wydajność kanału i zmniejszyć ruch w sieci. Za pomocą funkcji dostarczonej z produktem IBM MQmożna kompresować dane, które przepływają w kanałach komunikatów i kanałach MQI. W każdym z tych typów kanałów można kompresować dane nagłówka i dane komunikatów niezależnie od siebie. Domyślnie żadne dane nie są kompresowane w kanale.

Aplikacja IBM MQ classes for JMS określa techniki, które mogą być używane do kompresowania nagłówka lub danych 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, jest to kolejność, w jakiej techniki kompresji są negocjowane z menedżerem kolejek, gdy aplikacja tworzy połączenie. Aplikacja może następnie przekazać kolekcję do obiektu ConnectionFactory , wywołując metodę setHdrCompList(), dla danych nagłówka lub metodę setMsgCompList(), w celu uzyskania danych komunikatu. Gdy aplikacja jest gotowa, może utworzyć połączenie.

Opisane poniżej fragmenty kodu ilustrują opisane podejście. Pierwszy fragment kodu pokazuje, jak zaimplementować kompresję 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, kiedy to połączenie jest tworzone. Wybrana technika kompresji nie może zostać zmieniona w czasie życia obiektu połączenia. Aby można było używać kompresji w połączeniu, przed utworzeniem obiektu połączenia należy wywołać metodę setHdrCompList() i metody setMsgCompList().

Asynchronicznie umieszczanie komunikatów w produkcie IBM MQ classes for JMS
Zwykle, gdy aplikacja wysyła komunikaty do miejsca docelowego, aplikacja musi czekać na menedżera kolejek, aby upewnić się, że przetworzono żądanie. W niektórych sytuacjach wydajność przesyłania komunikatów można zwiększyć, wybierając opcję asynchronicznie umieszczając komunikaty. Gdy aplikacja umieszcza komunikat asynchronicznie, menedżer kolejek nie zwraca powodzenia lub niepowodzenia każdego wywołania, ale można okresowo sprawdzać, czy nie wystąpiły błędy.

Niezależnie od tego, czy miejsce docelowe zwraca sterowanie do aplikacji, bez określania, czy menedżer kolejek odebrał komunikat bezpiecznie, zależy od następujących właściwości:

Właściwość miejsca docelowego JMS PUTASYNCALLOWED (krótka nazwa-PAALD).
PUTASYNCALLOWED określa, czy aplikacje produktu JMS mogą umieszczać komunikaty asynchronicznie, jeśli ta opcja jest dozwolona przez kolejkę podstawową lub temat, który reprezentuje miejsce docelowe JMS .

Właściwość tematu lub kolejki produktu IBM MQ DEFPRESP (Domyślny typ odpowiedzi dla umieszczenia).

Funkcja DEFPRESP określa, czy aplikacje, które umieszczają komunikaty w kolejce, czy publikują komunikaty do tematu, mogą korzystać z funkcji put asynchronicznego.

W poniższej tabeli przedstawiono możliwe wartości dla właściwości PUTASYNCALLOWED i DEFPRESP oraz kombinacje wartości używanych do włączenia funkcji put asynchronicznego:

Tabela 45. Sposób łączenia właściwości PUTASYNCALLOWED i DEFPRESP w celu określenia, czy komunikaty są umieszczane w miejscu docelowym asynchronicznie.

Właściwość kolejki produktu IBM MQ	**PUTASYNCALLOWED = NIE**	**PUTASYNCALLOWED = TAK**	**PUTASYNCALLOWED = AS_DEST lub AS_Q_DEF lub AS_T_DEF**
DEFPRESP=SYNC	Funkcja asynchronicznej funkcji put nie została włączona	Włączona funkcja asynchronicznej funkcji put	Funkcja asynchronicznej funkcji put nie została włączona
DEFPRESP=ASYNC	Funkcja asynchronicznej funkcji put nie została włączona	Włączona funkcja asynchronicznej funkcji put	Włączona funkcja asynchronicznej funkcji put

W przypadku komunikatów wysyłanych w sesji transakcyjnych aplikacja ostatecznie określa, czy menedżer kolejek odebrał komunikaty w sposób bezpieczny, gdy wywołuje program `commit()`.

Jeśli aplikacja wysyła komunikaty trwałe w ramach sesji transakcyjnej, a co najmniej jeden komunikat nie jest odbierany bezpiecznie, transakcja nie zostanie zatwierdzona i zostanie zgłoszony wyjątek. Jeśli jednak aplikacja wysyła nietrwałe komunikaty w ramach sesji transakcyjnej, a co najmniej jeden komunikat nie zostanie odebrany bezpiecznie, transakcja zostanie pomyślnie wykonana. Aplikacja nie otrzymuje żadnych informacji zwrotnych, że komunikaty nietrwałe nie dotarły bezpiecznie.

W przypadku komunikatów nietrwałych wysłanych w sesji, która nie jest transakowana, właściwość SENDCHECKCOUNT obiektu *ConnectionFactory* określa liczbę komunikatów, które mają zostać wysłane, zanim program IBM MQ classes for JMS sprawdzi, czy menedżer kolejek odebrał komunikaty bezpiecznie.

Jeśli sprawdzenie wykryje, że co najmniej jeden komunikat nie został odebrany bezpiecznie, a aplikacja zarejestrowało obiekt nasłuchiwania wyjątków w połączeniu, produkt IBM MQ classes for JMS wywołuje metodę `onException()` obiektu nasłuchiwania wyjątków w celu przekazania wyjątku JMS do aplikacji.

Wyjątek JMS zawiera kod błędu JMSWMQ0028 , a 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. Wartość domyślna właściwości SENDCHECKCOUNT wynosi zero, co oznacza, że nie są wykonywane żadne sprawdzenia.

Ta optymalizacja jest najbardziej korzystna dla aplikacji, która łączy się z menedżerem kolejek w trybie klienta i wymaga wysyłania sekwencji komunikatów w szybkim dziedziczeniu, ale nie wymaga natychmiastowych informacji zwrotnych od menedżera kolejek dla każdego wysłanego komunikatu. Jednak aplikacja może nadal korzystać z tej optymalizacji nawet wtedy, gdy łączy się z menedżerem kolejek w trybie powiązań, ale oczekiwane korzyści z wydajności nie są tak duże.

Korzystanie z odczytu z wyprzedzeniem z produktem IBM MQ classes for JMS

Funkcja odczytu z wyprzedzeniem udostępniana przez produkt IBM MQ umożliwia wysyłanie nietrwałych komunikatów, które są odbierane poza transakcją, w celu wysłania ich do IBM MQ classes for JMS przed ich żądaniami. Program IBM MQ classes for JMS zapisuje komunikaty w buforze wewnętrznym i przekazuje je do aplikacji, gdy aplikacja zapyta o te komunikaty.

Aplikacje produktu 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. Funkcja odczytu z wyprzedzeniem umożliwia aplikacjom korzystaniowym z tych obiektów korzystanie z większej wydajności podczas odbierania komunikatów.

To, czy aplikacja, która używa produktu `MessageConsumers` lub `MessageListeners` , może używać odczytu z wyprzedzeniem, zależy od następujących właściwości:

Właściwość miejsca docelowego JMS READAHEADALLOWED (krótka nazwa-RAALD).
READAHEADALLOWED kontroluje, czy aplikacje JMS mogą korzystać z odczytu z wyprzedzeniem podczas pobierania lub przeglądania nietrwałych komunikatów poza transakcją, jeśli kolejka bazowa lub temat, który reprezentuje miejsce docelowe JMS , umożliwia tę opcję.

Kolejka IBM MQ lub właściwość tematu DEFREADA (domyślnie 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, a także kombinacje wartości używanych do włączania funkcji odczytu z wyprzedzeniem:

Tabela 46. W jaki sposób właściwości READAHEADALLOWED i DEFREADA są łączone w celu określenia, czy odczyt z wyprzedzeniem jest używany podczas odbierania lub przeglądania nietrwałych komunikatów poza transakcją.

Właściwość kolejki produktu IBM MQ	**READAHEADALLOWED = YES**	**READAHEADALLOWED = NIE**	**AS_DEST lub AS_Q_DEF lub AS_T_DEF**
DEFREADA = NIE	Funkcja odczytu z wyprzedzeniem włączona	Funkcja odczytu z wyprzedzeniem nie jest włączona	Funkcja odczytu z wyprzedzeniem nie jest włączona
DEFREADA = TAK	Funkcja odczytu z wyprzedzeniem włączona	Funkcja odczytu z wyprzedzeniem nie jest włączona	Funkcja odczytu z wyprzedzeniem włączona
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 `MessageConsumer` lub `MessageListener` jest tworzona przez aplikację, IBM MQ classes for JMS tworzy wewnętrzny bufor dla miejsca docelowego, które jest monitorowane przez produkt `MessageConsumer` lub `MessageListener` . Dla każdego `MessageConsumer` lub `MessageListener`istnieje jeden bufor wewnętrzny. Menedżer kolejek uruchamia wysyłanie nietrwałych komunikatów do partycji IBM MQ classes for JMS , gdy aplikacja wywołuje jedną z następujących metod:

- `MessageConsumer.receive()`
- `MessageConsumer.receive(long timeout)`
- `MessageConsumer.receiveNoWait()`
- `Session.setMessageListener(MessageListener listener)`

Program 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 utworzonym dla miejsca docelowego. Gdy aplikacja zażąda przetworzenia następnego komunikatu do przetworzenia, program IBM MQ classes for JMS zwróci następny komunikat w buforze wewnętrznym.

Jeśli bufor wewnętrzny jest pusty, program IBM MQ classes for JMS żąda bardziej nietrwałych komunikatów z menedżera kolejek.

Wewnętrzny bufor używany przez IBM MQ classes for JMS jest usuwany, gdy aplikacja zamknie `MessageConsumer`lub JMS Sesja, z którą powiązany jest `MessageListener` .

W przypadku systemu `MessageConsumers`wszystkie nieprzetworzone komunikaty w buforze wewnętrznym są tracone.

W przypadku korzystania z produktu `MessageListeners`, co dzieje się z komunikatami w buforze wewnętrznym, zależy od właściwości docelowej JMS READAHEADCLOSEPOLICY (krótka nazwa-RACP). Wartością domyślną właściwości jest DELIVER_ALL, co oznacza, że sesja JMS użyta do utworzenia partycji `MessageListener` nie jest zamknięta, dopóki wszystkie komunikaty w buforze wewnętrznym nie zostaną dostarczone do aplikacji. Jeśli właściwość jest ustawiona na DELIVER_CURRENT, sesja JMS zostanie zamknięta po przetworzeniu bieżącego komunikatu przez aplikację, a wszystkie pozostałe komunikaty w buforze wewnętrznym zostaną usunięte.

Zachowane publikacje w produkcie IBM MQ classes for JMS
Klient IBM MQ classes for JMS może być skonfigurowany do korzystania z zachowanych publikacji.

Publikator może określić, że kopia publikacji musi być zachowana, aby można ją było wysłać do przyszłych subskrybentów, którzy zarejestrują zainteresowanie tematem. Można to zrobić w programie IBM MQ classes for JMS , ustawiając właściwość całkowitoliczbową JMS_IBM_RETAIN na wartość 1. Stałe zostały zdefiniowane dla tych wartości w interfejsie com.ibm.msg.client.jms.JmsConstants . Na przykład, jeśli został utworzony 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 JMS_IBM_RETAIN może być również wysyłane w odebranym komunikacie. W związku z tym możliwe jest sprawdzenie, 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 produktu JEE .

Jeśli wymagane jest korzystanie z funkcji XA w środowisku serwera aplikacji, należy odpowiednio skonfigurować aplikację. Informacje na temat sposobu konfigurowania aplikacji do korzystania z transakcji rozproszonych można znaleźć w dokumentacji własnej serwera aplikacji.

Menedżer kolejek produktu IBM MQ nie może działać jako menedżer transakcji dla produktu JMS.

Korzystanie z funkcji produktu JMS 2.0
Produkt JMS 2.0 wprowadza kilka nowych obszarów funkcjonalności do produktu IBM MQ classes for JMS.

Podczas tworzenia aplikacji produktu JMS w wersji IBM MQ 8.0 lub nowszej może być konieczne rozważenie wpływu tej funkcjonalności na menedżer kolejek.

Pojęcia pokrewne
Interfejsy językowe programu IBM MQ Java

Opóźnienie dostarczania JMS 2.0
Za pomocą programu JMS 2.0można określić opóźnienie dostawy podczas wysyłania komunikatu. Menedżer kolejek nie dostarcza komunikatu do momentu, gdy upłynie określone opóźnienie dostawy.

Aplikacja może określić opóźnienie dostarczenia (w milisekundach), gdy wyśle komunikat, używając `MessageProducer.setDeliveryDelay(long deliveryDelay)` lub `JMSProducer.setDeliveryDelay(long deliveryDelay)`. Wartość ta jest dodawana do czasu, w którym komunikat jest wysyłany i podaje najwcześniejszy czas, w którym każda inna aplikacja może otrzymać ten komunikat.

W produkcie IBM MQ 8.0 i nowszych opóźnianie dostarczania jest implementowane przy użyciu pojedynczej wewnętrznej kolejki pomostowej. Komunikaty z niezerowym opóźnieniem dostarczenia są umieszczane w tej kolejce z nagłówkiem, który wskazuje opóźnienie dostawy i informacje o kolejce docelowej. Komponent menedżera kolejek, który jest wywoływany przez procesor opóźnienia dostarczania, monitoruje komunikaty w kolejce przemieszczania. Po zakończeniu opóźnienia dostarczania komunikatu komunikat zostaje wyłączony z kolejki przemieszczania i umieszczony w kolejce docelowej.

Klienty przesyłania komunikatów

Implementacja IBM MQ opóźnienia dostarczania jest dostępna tylko wtedy, gdy używany jest klient JMS . Jeśli opóźnienie dostawy jest używane z produktem IBM MQ, mają zastosowanie następujące ograniczenia. Ograniczenia te mają zastosowanie w równym stopniu do produktów `MessageProducers` i `JMSProducers`, ale `JMSRuntimeExceptions` są zgłaszane w przypadku produktu `JMSProducers`.

- Any attempt to call `MessageProducer.setDeliveryDelay` with a nonzero value when connected to a queue manager earlier than IBM MQ 8.0, results in a `JMSException` with a MQRC_FUNCTION_NOT_SUPPORTED message.
- Opóźnienie dostarczania nie jest obsługiwane dla miejsc docelowych w klastrze, które mają wartość **DEFBIND** inną niż MQBND_BIND_NOT_FIXED. Jeśli `MessageProducer` ma ustawioną niezerową opóźnienie dostarczania i zostanie podjęta próba wysłania do miejsca docelowego, które nie spełnia tego wymagania, to wywołanie spowoduje wywołanie `JMSException` z komunikatem MQRC_OPTIONS_ERROR.
- Każda próba ustawienia czasu na wartość życia, która jest mniejsza niż określone wcześniej niezerowe opóźnienie dostawy lub odwrotnie, powoduje, że w produkcie `JMSException` zostanie wyświetlony komunikat MQRC_EXPIRY_ERROR. Ta kontrola jest wykonywana w przypadku wywoływania metod `setTimeToLive` lub `setDeliveryDelay` lub `send` , w zależności od wybranego zestawu operacji.
- Korzystanie z zachowanych publikacji i opóźnianie dostawy nie jest obsługiwane. Próba opublikowania komunikatu z opóźnieniem dostawy, jeśli ten komunikat został oznaczony jako zachowany przez użycie produktu `msg.setIntProperty(JmsConstants.JMS_IBM_RETAIN, JmsConstants.RETAIN_PUBLICATION)` , spowoduje to `JMSException` z komunikatem MQRC_OPTIONS_ERROR.
- Opóźnienie dostawy i grupowanie komunikatów nie jest obsługiwane, a każda próba użycia tej kombinacji powoduje wystąpienie wyjątku JMSException z komunikatem MQRC_OPTIONS_ERROR.

Każde niepowodzenie wysłania komunikatu z opóźnieniem dostarczenia powoduje zgłoszenie klienta `JMSException` z odpowiednim komunikatem o błędzie, na przykład w kolejce zapełniania kolejki. W niektórych sytuacjach komunikat o błędzie może dotyczyć docelowego miejsca docelowego lub kolejki pomostowej lub obu tych elementów.

Uwaga: Program IBM MQ umożliwia aplikacjom umieszczanie komunikatów w jednostce pracy w celu ponownego uzyskania tego samego komunikatu, nawet jeśli jednostka pracy nie została zatwierdzona. Ta technika nie działa z opóźnieniem dostawy, ponieważ komunikat nie jest umieszczany w kolejce przemieszczania, dopóki jednostka pracy nie zostanie zatwierdzona, a wynik nie zostanie wysłany do docelowego miejsca docelowego.

Autoryzacja

Program IBM MQ przeprowadza sprawdzanie autoryzacji w oryginalnym docelowym miejscu docelowym, gdy aplikacja wysyła komunikat z niezerowym opóźnieniem dostawy. 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 dostawy.

- Multi W systemie Wiele platform, SYSTEM.DDELAY.LOCAL.QUEUE istnieje domyślnie. Kolejka systemowa musi być zmieniona, tak aby 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 dostawy zarówno do kolejek lokalnych, jak i współużytkowanych. W systemie z/OSnależy utworzyć kolejkę i zdefiniować ją w taki sposób, aby jej atrybuty MAXMSGL i MAXDEPTH były wystarczające dla oczekiwanego obciążenia.

Gdy ta kolejka jest tworzona, musi być ona zabezpieczona tak, aby jak najliczniej użytkownicy mieli do niej dostęp. Dostęp do kolejki musi być przeznaczony tylko do celów konserwacji i monitorowania.

Gdy 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 przemieszczania, jeśli jest to wymagane, na przykład w przypadku użycia dużego opóźnienia dostarczenia przez pomyłkę.

Uwagi dotyczące produktu z/OS

z/OS

Jeśli system działa w systemie z/OS, należy wziąć pod uwagę dodatkowe uwagi, które należy wziąć pod uwagę, jeśli chcesz użyć opóźnienia dostawy.

Jeśli ma być używane opóźnienie dostawy, kolejka systemowa SYSTEM.DDELAY.LOCAL.QUEUE musi być zdefiniowana. Musi być ona zdefiniowana z klasą pamięci wystarczającą dla oczekiwanego obciążenia oraz z określonymi wartością INDXTYPE (NONE) i MSGDLVSQ (FIFO). W pliku CSQ4INSG JCL została udostępniona przykładowa definicja kolejki systemowej, skomentowana z komentarzem.

Kolejki współużytkowane

Opóźnienie dostarczania jest obsługiwane w przypadku wysyłania komunikatów do współużytkowanych kolejek. Istnieje jednak tylko pojedyncza, prywatna kolejka pomostowa, która jest używana niezależnie od tego, czy kolejka docelowa jest współużytkowana, czy nie. Menedżer kolejek, który jest właścicielem tej kolejki prywatnej, musi być uruchomiony w celu wysłania opóźnionego komunikatu do docelowej kolejki docelowej po zakończeniu opóźnienia.

Uwaga: Jeśli komunikat nietrwały jest umieszczany z opóźnieniem dostarczenia do kolejki współużytkowanej, a menedżer kolejek, który jest właścicielem kolejki pomostowej, zostanie wyłączony, oryginalny komunikat zostanie utracony. Ponieważ komunikaty nietrwałe wysyłane z opóźnieniem dostawy do kolejki współużytkowanej są bardziej prawdopodobne, że zostaną utracone, niż komunikaty nietrwałe wysyłane bez opóźnienia dostarczania do współużytkowanej kolejki.

Docelowa rozdzielczość miejsca docelowego

Jeśli komunikat jest wysyłany do kolejki, rozdzielczość jest sterowana dwukrotnie; raz przez aplikację JMS i raz przez menedżer kolejek, gdy odbiera komunikat z kolejki przemieszczania i wysyła ją do kolejki docelowej.

Subskrypcje docelowe publikacji są dopasowywane, gdy aplikacja JMS wywołuje metodę wysyłania.

Jeśli komunikat jest wysyłany z trwałością lub priorytetem zgodnie z definicją kolejki, to wartość ta jest ustawiana w pierwszej rozdzielczości, a nie na drugiej.

Interwał utraty ważności

Opóźnienie dostarczania zachowuje zachowanie właściwości utraty ważności (MQMD.Expiry. Jeśli na przykład komunikat został umieszczony w aplikacji JMS z okresem ważności 20 000 ms, a opóźnienie dostarczenia 5000 ms i zostało wykonane po upłynięciu 10 000 ms, to wartość pola MQMD.expiry może wynosić około 50 dziesiątych sekundy. Wartość ta wskazuje, że upłynęło 15 sekund od momentu umieszczenia komunikatu, do czasu, kiedy został on wyświetlony.

Jeśli komunikat utraci ważność w czasie, gdy w kolejce pomostowej i w jednej z opcji MQRO_EXPIRATION_ * jest ustawiona, to wygenerowany raport jest dla pierwotnego komunikatu wysłanego przez aplikację, a nagłówek używany do przechowywania informacji o opóźnieniu dostawy jest usuwany.

Zatrzymywanie i uruchamianie procesora opóźnienia dostarczania

z/OS W systemie z/OSprocesor opóźnienia dostarczania jest zintegrowany z przestrzenią adresową MSTR menedżera kolejek. Gdy uruchamiany jest menedżer kolejek, uruchamiany jest również procesor opóźnienia dostarczania. Jeśli kolejka pomostowa jest dostępna, otwiera kolejkę i czeka na komunikaty, które mają zostać przetworzone. Jeśli kolejka pomostowa nie została zdefiniowana lub jest wyłączona lub wystąpi inny błąd, procesor opóźnienia dostawy zostanie wyłączony. Jeśli kolejka pomostowa jest w późniejszym czasie zdefiniowana lub zmieniona w celu włączenia, procesor opóźnia dostawę. Jeśli procesor opóźnienia dostawy zostanie wyłączony z innego powodu, można go zrestartować, zmieniając atrybut PUT kolejki pomostowej z ENABLED na DISABLED i ponownie z powrotem na ENABLED . Jeśli z jakiegokolwiek powodu konieczne jest zatrzymanie procesora opóźnienia dostarczania, należy ustawić atrybut PUT dla kolejki pomostowej na wartość DISABLED.

Multi W systemie Wiele platformprocesor opóźnienia rozpoczyna się od menedżera kolejek i jest automatycznie restartowany w przypadku wystąpienia błędu naprawialnego.

Niepowodzenie umieszczenia w kolejce docelowej

Jeśli opóźniony komunikat nie może zostać umieszczony w kolejce docelowej po zakończeniu opóźnienia, zostanie wyświetlony komunikat zgodnie z opcjami raportu: jest on usuwany lub wysyłany do kolejki niedostarczonego komunikatu. Jeśli to działanie nie powiedzie się, podejmowana jest próba późniejszego umieszczenia komunikatu. Jeśli działanie jest pomyślne, raport o wyjątku jest generowany i wysyłany do określonej kolejki, jeśli wymagane jest zgłoszenie raportu. Jeśli komunikat raportu nie może zostać wysłany, komunikat raportu jest wysyłany do kolejki niedostarczanej poczty. Jeśli wysyłanie raportu do kolejki niedostarczanych komunikatów nie powiedzie się, a komunikat jest trwały, wszystkie zmiany zostaną odrzucone, a oryginalny komunikat został wycofany i ponownie dostarczony. Jeśli komunikat nie jest trwały, komunikat raportu jest odrzucany, ale inne zmiany są zatwierdzane. Jeśli opóźniona publikacja nie może zostać dostarczona, ponieważ subskrybent nie zasubskrybował, lub w przypadku nietrwałego subskrybenta, ponieważ został rozłączony, wówczas komunikat zostanie odrzucony w trybie cichym. Komunikaty raportów są nadal generowane w sposób opisany wcześniej.

Jeśli opóźniona publikacja nie może zostać dostarczona do subskrybenta, a zamiast niej jest umieszczana w kolejce niedostarczonego komunikatu, a umieszczenie w kolejce niedostarczonego komunikatu nie powiedzie się, komunikat zostanie odrzucony.

Aby zmniejszyć prawdopodobieństwo niepowodzenia operacji umieszczania w kolejce docelowej po zakończeniu opóźnienia dostarczania, menedżer kolejek wykonuje pewne podstawowe operacje sprawdzania, gdy klient JMS wysyła komunikat z niezerowym opóźnieniem dostawy. Sprawdzają one, czy kolejka jest wyłączona, czy komunikat jest większy niż maksymalna dozwolona długość komunikatu, oraz czy kolejka jest pełna.

Publikowanie/subskrypcja

Dopasowywanie publikacji do dostępnych subskrypcji występuje, gdy aplikacja JMS wysyła komunikat z niezerowym opóźnieniem dostawy. Komunikat dla każdego zgodnego subskrybenta jest umieszczany w systemie SYSTEM.DDELAY.LOCAL.QUEUE , gdzie jest przechowywana do momentu zakończenia opóźnienia dostarczania. Jeśli jeden z tych subskrybentów jest subskrypcją proxy dla innego menedżera kolejek, to po zakończeniu opóźnienia dostarczania nastąpi przekroczenie czasu oczekiwania na ten menedżer kolejek. Może to spowodować, że subskrybenty w innym menedżerze kolejek odbierają publikacje, które zostały pierwotnie opublikowane przed subskrybentem. Jest to odchylenie od specyfikacji JMS 2.0 .

Opóźnienie dostarczania przy użyciu publikowania/subskrypcji jest obsługiwane tylko wtedy, gdy temat docelowy jest skonfigurowany z parametrem (N) PMSGDLV = ALLAVAIL. Próba użycia jakichkolwiek innych wartości powoduje błąd MQRC_PUBLICATION_FAILURE. Jeśli procesor opóźniania dostarczenia zakończy się niepowodzeniem podczas umieszczania komunikatu w kolejce docelowej, wynik jest opisany w sekcji Niepowodzenie umieszczenia w kolejce docelowej.

Komunikaty raportów

Wszystkie opcje raportów są obsługiwane i są obsługiwane przez procesor dostarczania, inne niż następujące opcje, które są ignorowane, ale przekazywane przez komunikat po wysłaniu do kolejki docelowej:

- MQRO_COA*
- MQRO_COD*
- MQRO_PAN/MQRO_NAN
- MQRO_ACTIVITY,

Klonowane i współużytkowane subskrypcje
W produkcie IBM MQ 8.0 lub nowszym istnieją dwie metody dające wielu konsumentom dostęp do tej samej subskrypcji. Te dwie metody są używane przy użyciu sklonowanych subskrypcji lub przy użyciu subskrypcji współużytkowanych.

Sklonowane subskrypcje

Sklonowana subskrypcja jest rozszerzeniem IBM MQ . Sklonowane subskrypcje umożliwiają wielu konsumentom na różnych Java maszynach wirtualnych (JVM) współbieżny dostęp do subskrypcji. To zachowanie może być używane przez ustawienie właściwości **CLONESUPP** na wartość `Włączona` w obiekcie connectionFactory . Domyślnie opcja **CLONESUPP** ma wartość `Wyłączone`. Sklonowane subskrypcje mogą być włączone tylko dla trwałych subskrypcji. Jeśli opcja **CLONESUPP** jest włączona, każde kolejne połączenie wykonywane przy użyciu tej opcji connectionFactory jest klonowane.

Subskrypcja trwała może zostać uznana za sklonowaną, jeśli co najmniej jeden konsument jest tworzony w celu odbierania komunikatów z tej subskrypcji, to znaczy zostały utworzone z podaniem tej samej nazwy subskrypcji. Można to zrobić tylko wtedy, gdy połączenie, w ramach którego utworzono konsumenty, ma wartość **CLONESUPP** ustawioną na wartość `Włączona` w obiekcie MQConnectionFactory. Po opublikowaniu komunikatu w temacie subskrypcji, kopia tego komunikatu jest wysyłana do subskrypcji. Komunikat jest dostępny dla każdego z konsumentów, ale tylko jeden z nich otrzymuje.

Uwaga: Włączenie klonowanych subskrypcji rozszerza specyfikację produktu JMS .

Subskrypcje współużytkowane

Specyfikacja JMS 2.0 wprowadza subskrypcje współużytkowane, które umożliwiają współużytkowanie komunikatów z subskrypcji tematu wśród wielu konsumentów. Każdy komunikat z subskrypcji jest dostarczany tylko do jednego z konsumentów w tej subskrypcji. Subskrypcje współużytkowane są włączane przez odpowiednie wywołanie funkcji API produktu JMS 2.0 .

Funkcje API mogą być wywoływane w jeden z następujących sposobów:

- Z poziomu aplikacji Java SE (lub kontenera klienta Java EE).
- Z serwletu lub implementacji komponentu MDB.

Specyfikacja JMS 2.0 nie definiuje żadnych standardowych sposobów prowadzenia komponentu MDB z poziomu sharedSubscription, dlatego produkt IBM MQ 8.0 lub nowszy udostępnia właściwość specyfikacji aktywowania sharedSubscription w tym celu. Więcej informacji na temat tej właściwości można znaleźć w sekcji “Konfigurowanie adaptera zasobów na potrzeby komunikacji przychodzącej” na stronie 458 i “Przykłady definiowania właściwości sharedSubscription” na stronie 476.

Jeśli subskrypcja współużytkowana jest włączona, nie może być niewspółużytkowana.

Subskrypcje współużytkowane mogą być tworzone jako subskrypcje trwałe lub nietrwałe. Nie ma wymogu, aby oddzielnie tworzyć obiekty po stronie menedżera kolejek wykraczające poza normalną konfigurację produktu JMS . Obiekty, które są wymagane, są tworzone dynamicznie.

Podejmowanie decyzji między współużytkowanymi lub sklonowanymi subskrypcjami

Podczas określania, czy mają być używane subskrypcje współużytkowane lub sklonowane, należy wziąć pod uwagę korzyści wynikające z obu tych opcji. Jeśli to możliwe, należy użyć subskrypcji współużytkowanych, ponieważ jest to zachowanie zdefiniowane w specyfikacji, a nie rozszerzenie specyficzne dla produktu IBM MQ .

Poniższa tabela zawiera kilka punktów, które należy wziąć pod uwagę podczas podejmowania decyzji między subskrypcjami współużytkowanymi i klonowanymi:

Tabela 47. Uwagi dotyczące wyboru między subskrypcjami współużytkowanymi i klonowanymi subskrypcjami

Subskrypcje współużytkowane	**Sklonowane subskrypcje**
Subskrypcje współużytkowane to standardowa część specyfikacji produktu JMS 2.0 .	Sklonowane subskrypcje są rozszerzeniem specyficznym dla produktu IBM MQ .
Subskrypcje współużytkowane są tworzone za pomocą jawnych wywołań metod API.	Sklonowane subskrypcje są sterowane administracyjnie na poziomie ConnectionFactory .
Subskrypcje współużytkowane mogą być trwałe lub nietrwałe.	Sklonowane subskrypcje mogą być trwałe.
Subskrypcje współużytkowane są jawnie tworzone w ramach subskrypcji indywidualnej.	Sklonowane subskrypcje są używane dla każdej trwałej subskrypcji w ramach połączenia, dla którego ta funkcja jest włączona.
Jeśli subskrypcja została utworzona jako współużytkowana, nie można jej później zmienić na niewspółużytkowaną lub odwrotnie.	Subskrypcja może zostać zmieniona z klonowania na niesklonowany za każdym razem, gdy zostanie ponownie otwarta, jeśli właściwość **CLONESUPP** będącego właścicielem połączenia została zmieniona.

Pojęcia pokrewne

Subskrybenty i subskrypcje

Trwałość subskrypcji

Zadania pokrewne

Korzystanie z subskrypcji współużytkowanych JMS 2.0

Odsyłacze pokrewne

“Przykłady definiowania właściwości sharedSubscription” na stronie 476

Właściwość sharedSubscription specyfikacji aktywowania można zdefiniować w pliku WebSphere Liberty `server.xml` . Alternatywnie można zdefiniować właściwość w komponencie bean sterowanym komunikatami (MDB) za pomocą adnotacji.

CLONESUPP

Właściwość SupportMQExtensions

Specyfikacja JMS 2.0 wprowadza zmiany w sposobie pracy niektórych zachowań. W produkcie IBM MQ 8.0 i nowszych znajduje się właściwość `com.ibm.mq.jms.SupportMQExtensions`, którą można ustawić na wartość *TRUE*, aby przywrócić te zmienione zachowania z powrotem do poprzednich implementacji.

Trzy obszary funkcjonalności są odwracane przez ustawienie parametru `SupportMQExtensions` na wartość *True*:

Priorytet komunikatu

Do wiadomości można przypisać priorytet, *0* - *9*. Przed JMS 2.0komunikaty mogą również używać wartości *-1*, co wskazuje, 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, które nie mają wartości NULL, były sprawdzane pod kątem unikalności podczas nawiązywania połączenia. Włączenie produktu `SupportMQExtensions`oznacza, że wymaganie to nie jest brane pod uwagę i że identyfikator klienta może zostać ponownie wykorzystany.

NoLocal
Specyfikacja JMS 2.0 wymaga, aby po włączeniu tej stałej konsument nie mógł odbierać komunikatów publikowanych przez ten sam identyfikator klienta. Przed JMS 2.0ten atrybut został ustawiony w subskrybencie, aby uniemożliwić odbieranie komunikatów publikowanych przez jego własne połączenie. Włączenie produktu `SupportMQExtensions` powoduje przywrócenie tego działania do jego poprzedniej implementacji.

Właściwość `com.ibm.mq.jms.SupportMQExtensions` to właściwość boolowska zawarta w produkcie `com.ibm.mqjms.jar`. 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 znajdującą się w pliku konfiguracyjnym produktu IBM MQ classes for JMS .

Pojęcia pokrewne
“Plik konfiguracyjny IBM MQ classes for JMS” na stronie 95
Plik konfiguracyjny IBM MQ classes for JMS określa właściwości, które są używane do konfigurowania produktu IBM MQ classes for JMS.

Odsyłacze pokrewne
“Właściwości używane do konfigurowania zachowania klienta JMS” na stronie 102
Za pomocą tych właściwości można skonfigurować zachowanie klienta JMS .

Korzystanie z subskrypcji współużytkowanych w produkcie JMS 2.0
Produkt JMS 2.0 wprowadza pojęcie produktu `shared subscriptions`, w którym pojedyncza subskrypcja jest współużytkowana przez wielu konsumentów, przy czym tylko jeden z konsumentów otrzymuje publikację w dowolnym momencie. IBM MQ classes for JMS.

Podczas tworzenia aplikacji produktu JMS w wersji IBM MQ 8.0 lub nowszej może być konieczne rozważenie wpływu tej funkcjonalności na menedżer kolejek.

Pomysł, który stoi za subskrypcjami współużytkowanymi, polega na tym, że obciążenie jest współużytkowane przez wielu konsumentów. Trwała subskrypcja może być również współużytkowana przez wielu konsumentów.

Załóżmy na przykład, że istnieje:

- Subskrypcja SUB, subskrybująca temat FIFA2014/UPDATES , aby otrzymywać aktualizacje zgodne z piłką nożną, są współużytkowane przez trzech konsumentów C1, C2i C3
- Publikowanie informacji o producencie P1 w temacie FIFA2014/UPDATES

Po opublikowaniu publikacji FIFA2014/UPDATESpublikacja będzie otrzymana tylko przez jednego z trzech konsumentów (C1, C2lub C3), ale nie wszystkie.

W poniższym przykładzie przedstawiono użycie współużytkowanych subskrypcji, a także demonstruje użycie dodatkowego interfejsu API w usłudze JMS 2.0 `Message.receiveBody()`w celu pobrania tylko treści komunikatu.

W tym przykładzie tworzone są trzy wątki subskrybenta, które tworzą współużytkowaną subskrypcję tematu FIFA2014/UPDATES i jednego wątku publikatora.

```
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
    */
    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 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 Simplified API using IBM MQ v91 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 językowe programu IBM MQ Java

Narzędzia serwera aplikacji IBM MQ classes for JMS

W tym temacie opisano, w jaki sposób produkt IBM MQ classes for JMS implementuje klasę ConnectionConsumer i zaawansowaną funkcjonalność w klasie sesji. Podsumowuje on również funkcję puli sesji serwera.

Ważne: Niniejsza informacja dotyczy wyłącznie odniesienia. Aplikacja nie może być zapisana w celu użycia tego interfejsu: jest ona używana w adapterze zasobów produktu IBM MQ do nawiązywania połączenia z serwerami Java EE. Informacje o połączeniu praktycznym znajdują się w sekcji “Korzystanie z adaptera zasobów IBM MQ” na stronie 442.

Produkt IBM MQ classes for JMS obsługuje narzędzia Application Server Facilities (ASF) określone w specyfikacji *Java Message Service Specification* (patrz sekcja Oracle Technology Network for Java Developers). Ta specyfikacja identyfikuje trzy role w ramach tego modelu programowania:

- **Dostawca produktu JMS** udostępnia funkcje ConnectionConsumer i zaawansowane funkcje sesji.
- **Serwer aplikacji** udostępnia funkcje puli ServerSessionoraz funkcji ServerSession .
- **Aplikacja kliencka** korzysta z funkcji dostarczanej przez dostawcę JMS i serwer aplikacji.

Informacje zawarte w tym temacie nie mają zastosowania, jeśli aplikacja korzysta z połączenia w czasie rzeczywistym z brokerem.

Obiekt JMS ConnectionConsumer

Interfejs ConnectionConsumer udostępnia wysokowydajną metodę dostarczania komunikatów współbieżnie do puli wątków.

Specyfikacja JMS umożliwia serwerom aplikacji ścisłą integrację z implementacją serwera JMS przy użyciu interfejsu `ConnectionConsumer` . Ten składnik udostępnia współbieżne przetwarzanie komunikatów. Zwykle serwer aplikacji tworzy pulę wątków, a implementacja JMS udostępnia komunikaty do tych wątków. Serwer aplikacji z rozpoznaniem JMS(taki jak WebSphere Application Server) może

korzystać z tej funkcji w celu udostępnienia funkcji przesyłania komunikatów wysokiego poziomu, takich jak komponenty bean sterowane komunikatami.

W normalnych aplikacjach nie jest używany obiekt ConnectionConsumer, ale mogą być używane przez klienty JMS eksperckie. W przypadku takich klientów ConnectionConsumer udostępnia wysokowydajną metodę dostarczania komunikatów współbieżnie do puli wątków. Gdy komunikat dociera do kolejki lub tematu, program JMS wybiera wątek z puli i dostarcza do niego zadania wsadowe. W tym celu produkt JMS uruchamia powiązaną metodę `onMessage()` obiektu MessageListener.

Ten sam efekt można osiągnąć, konstruując wiele obiektów sesji i obiektów MessageConsumer , a każdy z nich ma zarejestrowaną wartość MessageListener. Jednak ConnectionConsumer zapewnia lepszą wydajność, mniejsze wykorzystanie zasobów i większą elastyczność. W szczególności wymagane jest mniej obiektów sesji.

Planowanie aplikacji z ASF

W tej sekcji opisano, jak zaplanować aplikację, w tym:

- "Ogólne zasady przesyłania komunikatów w trybie punkt z punktem przy użyciu narzędzia ASF" na stronie 329
- "Ogólne zasady przesyłania komunikatów w trybie publikowania/subskrypcji przy użyciu narzędzia ASF" na stronie 330
- "Usuwanie komunikatów z kolejki w ASF" na stronie 331
- Obsługa komunikatów nieprzetwarzalnych w ASF. Patrz "Obsługa komunikatów nieprzetwarzalnych w produkcie IBM MQ classes for JMS" na stronie 223.

Ogólne zasady przesyłania komunikatów w trybie punkt z punktem przy użyciu narzędzia ASF
Ten temat zawiera ogólne informacje na temat przesyłania komunikatów w trybie punkt z punktem przy użyciu narzędzia ASF.

Gdy aplikacja tworzy obiekt ConnectionConsumer z obiektu QueueConnection , określa on obiekt kolejki produktu JMS oraz łańcuch selektora. Następnie element ConnectionConsumer rozpoczyna udostępnianie komunikatów do sesji w powiązanej puli ServerSession. Komunikaty docierają do kolejki, a jeśli są zgodne z selektorem, są dostarczane do sesji w powiązanej puli ServerSession.

W terminach IBM MQ obiekt kolejki odwołuje się do QLOCAL lub QALIAS w lokalnym menedżerze kolejek. Jeśli jest to QALIAS, to QALIAS musi odwoływać się do QLOCAL. W pełni rozwiązana wartość IBM MQ QLOCAL jest znana jako *bazowa wartość QLOCAL*. Parametr ConnectionConsumer ma wartość *active* (aktywny), jeśli nie jest zamknięty, a jego element nadrzędny QueueConnection jest uruchomiony.

Możliwe jest, aby wiele ConnectionConsumers, każdy z różnymi selektorami, było uruchamiane względem tego samego bazowego systemu QLOCAL. Aby zachować wydajność, niepożądane komunikaty nie mogą być gromadzone w kolejce. Niechciane komunikaty to te, dla których żaden aktywny element ConnectionConsumer nie ma zgodnego selektora. Fabrykę QueueConnectionmożna ustawić w taki sposób, aby niepożądane komunikaty zostały usunięte z kolejki (szczegółowe informacje na ten temat zawiera sekcja "Usuwanie komunikatów z kolejki w ASF" na stronie 331). To zachowanie można ustawić na jeden z dwóch sposobów:

- Za pomocą narzędzia administracyjnego JMS ustaw fabrykę QueueConnectionna wartość MRET (NO).
- W programie należy użyć:

```
MQQueueConnectionFactory.setMessageRetention(WMQConstants.WMQ_MRET_NO)
```

Jeśli to ustawienie nie zostanie zmienione, wartością domyślną będzie zachowanie takich niechcianych komunikatów w kolejce.

Podczas konfigurowania menedżera kolejek produktu IBM MQ należy wziąć pod uwagę następujące kwestie:

- Dla danych wejściowych współużytkowanych musi być włączona bazowa wartość QLOCAL. Aby to zrobić, należy użyć następującej komendy MQSC:

```
ALTER QLOCAL( your.qlocal.name ) SHARE GET(ENABLED)
```

- Menedżer kolejek musi mieć włączoną kolejkę niedostarczonych komunikatów. Jeśli obiekt ConnectionConsumer wystąpi z problemem, gdy umieszcza komunikat w kolejce niedostarczonych komunikatów, dostarczenie komunikatu od bazowego zatrzymania QLOCAL zostanie zatrzymane. Aby zdefiniować kolejkę niedostarczonych komunikatów, należy użyć następującej komendy:

```
ALTER QMGR DEADQ( your.dead.letter.queue.name )
```

- Użytkownik, który uruchamia obiekt 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 produktu IBM MQ dla konkretnej platformy.
- Jeśli w kolejce pozostawiane są niechciane komunikaty, pogarszają one wydajność systemu. Dlatego należy zaplanować selektory komunikatów w taki sposób, aby między nimi ConnectionConsumers usunował wszystkie komunikaty z kolejki.

Szczegółowe informacje na temat komend MQSC zawiera sekcja Komendy MQSC.

Ogólne zasady przesyłania komunikatów w trybie publikowania/subskrypcji przy użyciu narzędzia ASF
ConnectionConsumers odbiera komunikaty dla określonego tematu. Element ConnectionConsumer może być trwały lub nie jest trwały. Należy określić kolejkę lub kolejki, których używa ConnectionConsumer .

Gdy aplikacja tworzy obiekt ConnectionConsumer z obiektu TopicConnection , określa on obiekt tematu i łańcuch selektora. Następnie element ConnectionConsumer rozpoczyna odbieranie komunikatów zgodnych z selektorem w tym temacie, w tym wszystkie zachowane publikacje dotyczące subskrybowanego tematu.

Alternatywnie aplikacja może utworzyć trwałą wartość ConnectionConsumer , która jest powiązana z konkretną nazwą. Ten obiekt ConnectionConsumer odbiera komunikaty, które zostały opublikowane w ramach tematu, ponieważ trwały obiekt ConnectionConsumer był ostatnio aktywny. Odbiera on wszystkie takie komunikaty, które są zgodne z selektorem w temacie. Jeśli jednak obiekt ConnectionConsumer korzysta z funkcji odczytu z wyprzedzeniem, może utracić nietrwałe komunikaty, które znajdują się w buforze klienta po jego zamknięciu.

Jeśli produkt IBM MQ classes for JMS znajduje się w trybie migracji dostawcy przesyłania komunikatów produktu IBM MQ , dla nietrwałych subskrypcji ConnectionConsumer używana jest osobna kolejka.Konfigurowalna opcja CCSUB w fabryce TopicConnectionokreśla kolejkę, która ma być używana. Normalnie CCSUB określa pojedynczą kolejkę do użycia przez wszystkie ConnectionConsumers , które korzystają z tej samej fabryki TopicConnection. Jednak możliwe jest, aby każdy obiekt ConnectionConsumer wygenerował kolejkę tymczasową, określając przedrostek nazwy kolejki, po którym następuje gwiazdka (*).

Jeśli produkt IBM MQ classes for JMS znajduje się w trybie migracji dostawcy przesyłania komunikatów produktu IBM MQ , właściwość CCDSUB tematu określa kolejkę, która ma być używana dla trwałych subskrypcji. Ponownie może to być kolejka, która już istnieje, lub przedrostek nazwy kolejki, po którym następuje gwiazdka (*). Jeśli zostanie określona kolejka, która już istnieje, wszystkie trwałe elementy ConnectionConsumers , które subskrybują ten temat, używają tej kolejki. Jeśli zostanie określony przedrostek nazwy kolejki, po którym następuje gwiazdka (*), kolejka jest generowana po raz pierwszy, gdy zostanie utworzona trwała wartość ConnectionConsumer o określonej nazwie. Ta kolejka jest ponownie wykorzystywana później, gdy zostanie utworzona trwała wartość ConnectionConsumer o tej samej nazwie.

Podczas konfigurowania menedżera kolejek produktu IBM MQ należy wziąć pod uwagę następujące kwestie:

- Menedżer kolejek musi mieć włączoną kolejkę niedostarczonych komunikatów. Jeśli obiekt ConnectionConsumer wystąpi z problemem, gdy umieszcza komunikat w kolejce niedostarczonych komunikatów, dostarczenie komunikatu od bazowego zatrzymania QLOCAL zostanie zatrzymane. Aby zdefiniować kolejkę niedostarczonych komunikatów, należy użyć następującej komendy:

```
ALTER QMGR DEADQ( your.dead.letter.queue.name )
```

- Użytkownik, który uruchamia obiekt ConnectionConsumer , musi mieć uprawnienia do wykonywania operacji MQOPEN z MQOO_SAVE_ALL_CONTEXT i MQOO_PASS_ALL_CONTEXT. Szczegółowe informacje można znaleźć w dokumentacji produktu IBM MQ dla używanej platformy.
- Wydajność pojedynczego obiektu ConnectionConsumer można zoptymalizować, tworząc osobną, dedykowaną dla niego kolejkę. Jest to koszt dodatkowego wykorzystania zasobów.

Usuwanie komunikatów z kolejki w ASF
Gdy aplikacja korzysta z opcji ConnectionConsumers, może być konieczne usunięcie komunikatów z kolejki w wielu sytuacjach przez produkt JMS .

Są to następujące sytuacje:

Źle sformatowany komunikat
Może dojść do sytuacji, w której produkt JMS nie może przeanalizować składni.

Wiadomość nieprzetwarzalna
Komunikat może osiągnąć próg wycofania, ale element ConnectionConsumer nie może ponownie umieścić go w kolejce wycofanych komunikatów.

Brak zainteresowania ConnectionConsumer
W przypadku przesyłania komunikatów w trybie punkt z punktem, gdy fabryka QueueConnectionjest ustawiona w taki sposób, że nie zachowuje niechcianych komunikatów, pojawia się komunikat niepożądany przez dowolny z elementów ConnectionConsumers(Złącza połączenia).

W takich sytuacjach element ConnectionConsumer próbuje usunąć komunikat z kolejki. Opcje rozporządzania w polu raportu deskryptora MQMD komunikatu ustawiają dokładne zachowanie. Są to następujące opcje:

MQRO_DEAD_LETTER_Q
Komunikat ten jest ponownie wysyłany do kolejki niedostarczonych komunikatów menedżera kolejek. Jest to opcja domyślna.

MQRO_DISCARD_MSG
Komunikat jest odrzucany.

Opcja ConnectionConsumer generuje również komunikat raportu, a to zależy również od pola raportu MQMD komunikatu. Ten komunikat jest wysyłany do kolejki ReplyTokomunikatu w menedżerze kolejek ReplyTo. Jeśli podczas wysyłania komunikatu o raporcie wystąpi błąd, komunikat jest wysyłany do kolejki niedostarczonych komunikatów. Opcje raportu o wyjątkach w polu raportu w szczegółach MQMD komunikatu ustawiają szczegóły komunikatu raportu. Są to następujące opcje:

MQRO_EXCEPTION
Generowany jest 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.

Gdy generowane są komunikaty raportów, uhonorowane są następujące opcje:

- MQRO_NEW_MSG_ID
- MQRO_PASS_MSG_ID
- MQRO_COPY_MSG_ID_TO_CORREL_ID (Identyfikator CORREL_ID)
- MQRO_PASS_CORREL_ID

Jeśli komunikat nieprzetwarzalny nie może być ponownie wysłany, być może dlatego, że kolejka niedostarczonych komunikatów jest pełna lub autoryzacja została błędnie określona, 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 na tym miejscu docelowym jest zatrzymywane. Tego typu połączenia muszą zostać zamknięte, a problem rozwiązany, zanim możliwe będzie ponowne utworzenie i ponowne uruchomienie dostarczania komunikatów.

Ważne jest, aby zdefiniować kolejkę niedostarczonych komunikatów i regularnie sprawdzać, czy nie występują żadne problemy. W szczególności zadbaj o to, aby kolejka niedostarczonych komunikatów nie osiągnęła maksymalnej głębokości, a jej maksymalna wielkość komunikatu jest wystarczająco duża dla wszystkich komunikatów.

W przypadku, gdy komunikat jest ponownie wysyłany do kolejki niedostarczonych komunikatów, jest on poprzedzony nagłówkiem IBM MQ niedostarczonych komunikatów (MQDLH). Szczegółowe informacje na temat formatu MQDLH można znaleźć w sekcji MQDLH-Dead-letter header (Nagłówek niewysłanych wiadomości MQDLH). Użytkownik może zidentyfikować komunikaty, które ConnectionConsumer umieli w kolejce niedostarczonych komunikatów, lub komunikaty raportów, które wygenerował ConnectionConsumer , za pomocą następujących pól:

- PutApplTyp: `MQAT_JAVA (0x1C)`
- PutApplNazwa: "`MQ JMS ConnectionConsumer`"

Pola te znajdują się w komunikatach MQDLH komunikatów w kolejce niedostarczonych komunikatów oraz w deskrypcie MQMD komunikatów raportu. Pole informacji zwrotnej deskryptora MQMD oraz pole Przyczyna komunikatu MQDLH zawiera kod opisujący błąd. Szczegółowe informacje na temat tych kodów zawiera sekcja “Kody przyczyny i sprzężenia zwrotnego w ASF” na stronie 333. Inne pola są opisane w sekcji Nagłówek MQDLH-Dead-letter.

Obsługa komunikatów nieprzetwarzalnych w ASF
W obiektach serwera aplikacji obsługa komunikatów nieprzetwarzalnych jest nieco inna niż w innych miejscach w produkcie IBM MQ classes for JMS.

Więcej informacji 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 223.

Jeśli używane są narzędzia Application Server Facilities (ASF), ConnectionConsumer, a nie MessageConsumer, przetwarza komunikaty nieprzetwarzalne. Właściwości ConnectionConsumer są requiem komunikatów zgodnie z właściwościami QName BackoutThreshold i BackoutRequeuekolejki.

Jeśli aplikacja korzysta z opcji ConnectionConsumers, okoliczności, w których jest tworzona kopia zapasowa komunikatu, zależą od sesji, którą udostępnia serwer aplikacji:

- Gdy sesja jest nietransakowana, z AUTO_ACKNOWLEDGE lub DUPS_OK_ACKNOWLEDGE, komunikat jest wycofany tylko po wystąpieniu błędu systemowego lub nieoczekiwanie kończy działanie aplikacji.
- Gdy sesja nie jest transakowana za pomocą funkcji CLIENT_ACKNOWLEDGE, niepotwierdzone komunikaty mogą być wycofane przez serwer aplikacji wywołując komendę Session.recover().

  Zazwyczaj implementacja klienta MessageListener lub serwer aplikacji wywołuje metodę Message.acknowledge(). Plik Message.acknowledge() potwierdza wszystkie komunikaty dostarczone do tej pory w sesji.
- Gdy sesja jest transacted, niepotwierdzone komunikaty mogą być wycofane przez serwer aplikacji wywołując komendę Session.rollback().
- Jeśli serwer aplikacji dostarcza XASession, komunikaty są zatwierdzane lub wycofane w zależności od transakcji rozproszonej. Serwer aplikacji bierze odpowiedzialność za wykonanie transakcji.

Pojęcia pokrewne
“Obsługa komunikatów nieprzetwarzalnych w produkcie IBM MQ classes for JMS” na stronie 223

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

Ta sekcja obejmuje różne aspekty obsługi błędów, w tym produkty "Odtwarzanie po wystąpieniu błędów w ASF" na stronie 333 i "Kody przyczyny i sprzężenia zwrotnego w ASF" na stronie 333.

Odtwarzanie po wystąpieniu błędów w ASF
Jeśli ConnectionConsumer doświadcza poważnego błędu, dostarczenie komunikatu do wszystkich elementów ConnectionConsumers z zainteresowaniem w tych samych zatrzymań QLOCAL. W takiej sytuacji każdy obiekt ExceptionListener , który jest zarejestrowany w dotkniętym połączeniu, jest powiadamiany o tym. Istnieją dwa sposoby, w których aplikacja może odtworzyć dane z tych warunków.

Zwykle poważny błąd o tej naturze występuje wtedy, gdy ConnectionConsumer nie może przekwalifikować komunikatu do kolejki niedostarczonych komunikatów, albo wystąpi błąd podczas odczytywania komunikatów z kolejki QLOCAL.

Ponieważ każdy obiekt ExceptionListener , który jest zarejestrowany w dotkniętym połączeniu, jest powiadamiany, można użyć ich w celu zidentyfikowania przyczyny problemu. W niektórych przypadkach administrator systemu musi interweniować, aby rozwiązać ten problem.

Użyj jednej z następujących technik, aby odtworzyć dane z następujących warunków błędu:

- Wywołaj komendę `close()` dla wszystkich ConnectionConsumers. Aplikacja może utworzyć nowe ConnectionConsumers tylko wtedy, gdy wszystkie ConnectionConsumers zostaną zamknięte, a wszystkie problemy z systemem zostaną rozwiązane.
- Wywołaj `stop()` dla wszystkich połączeń, których dotyczy problem. Po zatrzymaniu wszystkich połączeń i rozwiązaniu ewentualnych problemów z systemem aplikacja może pomyślnie `start()` jej połączenia.

Kody przyczyny i sprzężenia zwrotnego w ASF
Użyj kodów przyczyny i informacji zwrotnych, aby określić przyczynę błędu. W tym miejscu podano wspólne kody przyczyny wygenerowane przez ConnectionConsumer .

Aby określić przyczynę błędu, należy użyć następujących informacji:

- Kod opinii we wszystkich komunikatach raportu
- Kod przyczyny w MQDLH wszystkich komunikatów w kolejce niedostarczonych komunikatów.

ConnectionConsumers generuje następujące kody przyczyny.

MQRC_BACKOUT_THRESHOLD_REACHED (0x93A; 2362)

Przyczyna
Komunikat osiągnął próg wycofania zdefiniowany w systemie QLOCAL, ale nie zdefiniowano kolejki wycofanych komunikatów.

Na platformach, na których nie można zdefiniować kolejki wycofanych komunikatów, komunikat osiągnął próg wycofania zdefiniowany przez JMS, który wynosi 20.

Działanie
Jeśli nie jest to wymagane, zdefiniuj kolejkę wycofania dla odpowiedniej kolejki QLOCAL. Poszukaj także przyczyny wielu wycofań.

MQRC_MSG_NOT_MATCHED (0x93B; 2363)

Przyczyna
W przesyłaniu komunikatów w trybie punkt z punktem istnieje komunikat, który nie jest zgodny z żadnym z selektorów dla elementu ConnectionConsumers , który monitoruje kolejkę. Aby zachować wydajność, komunikat jest ponownie wysyłany do kolejki niedostarczonych komunikatów.

Działanie
Aby uniknąć takiej sytuacji, należy upewnić się, że opcja ConnectionConsumers przy użyciu kolejki udostępnia zestaw selektorów, które zajmują się wszystkimi komunikatami, lub ustaw fabrykę QueueConnectionw celu zachowania komunikatów.

Można również zbadać źródło komunikatu.

MQRC_JMS_FORMAT_ERROR (0x93C; 2364)

Przyczyna
Program JMS nie może zinterpretować komunikatu w kolejce.

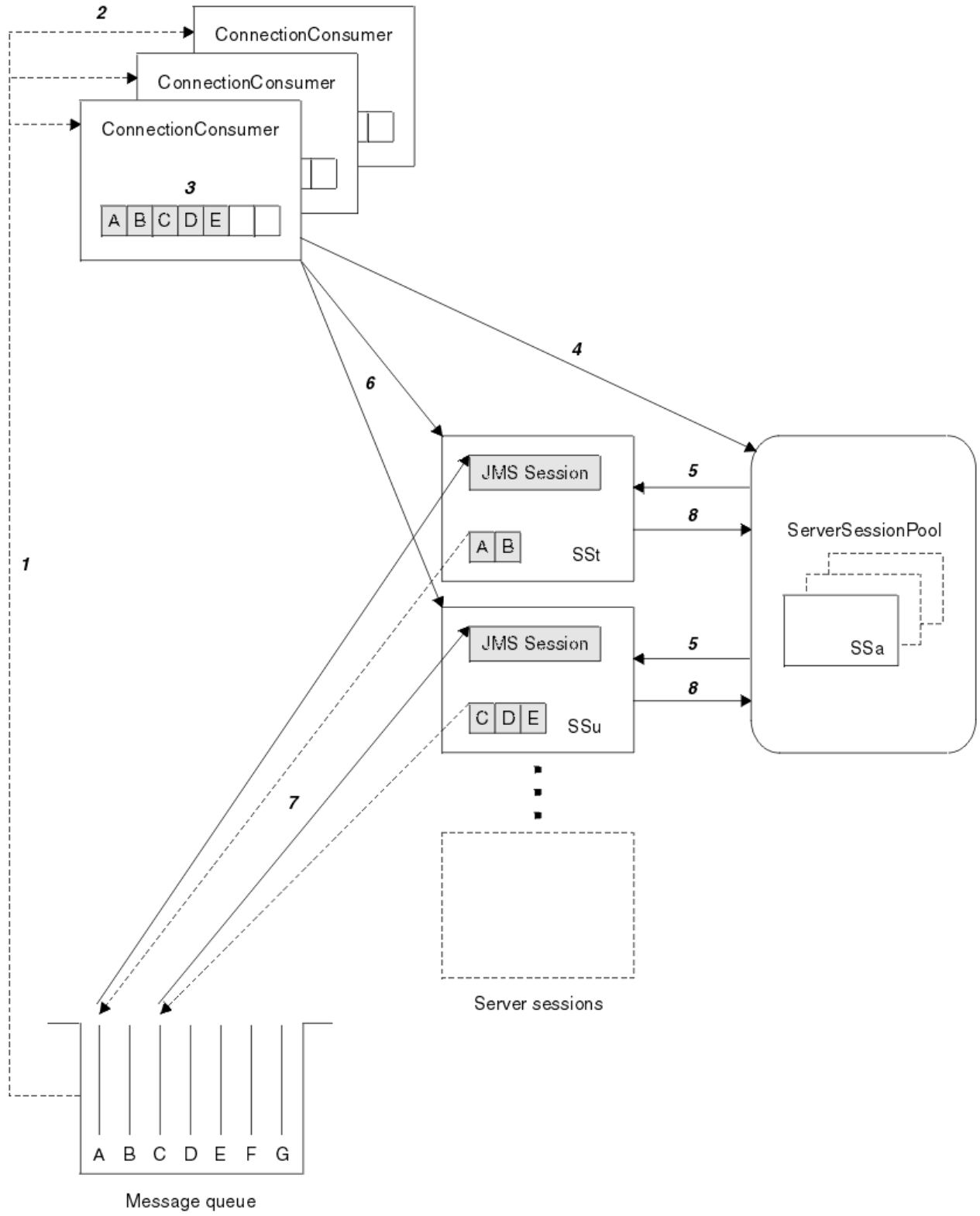
Działanie
Sprawdź pochodzenie komunikatu. Produkt JMS zwykle dostarcza komunikaty o nieoczekiwanym formacie w postaci BytesMessage lub TextMessage. Zdarza się, że błąd ten nie powiedzie się, jeśli komunikat jest bardzo źle sformatowany.

Inne kody, które pojawiają się w tych polach, są spowodowane przez nieudane próby ponownego umieszczenia komunikatu w kolejce wycofania. W tej sytuacji kod opisuje przyczynę niepowodzenia ponownego wykonania żądania. Aby zdiagnozować przyczynę tych błędów, należy zapoznać się z kodami zakończenia i przyczyny interfejsu API.

Jeśli komunikat raportu nie może zostać umieszczony w kolejce ReplyTo, jest on umieszczany w kolejce niedostarczonych komunikatów. W tej sytuacji pole sprzężenia zwrotnego deskryptora MQMD zostało zakończone zgodnie z opisem w tym temacie. Pole przyczyny w tabeli MQDLH wyjaśnia, dlaczego komunikat raportu nie może zostać umieszczony w kolejce ReplyTo.

Funkcja puli sesji serwera w systemie AFS
W tej sekcji przedstawiono podsumowanie funkcji puli sesji serwera.

Rysunek 51 na stronie 335 podsumowuje zasady działania puli ServerSessioni funkcji ServerSession .



Rysunek 51. Funkcje ServerSessionPool i ServerSession

1. Element ConnectionConsumers umożliwia pobranie odwołań do komunikatów z kolejki.
2. Każdy element ConnectionConsumer wybiera konkretne odwołania do komunikatów.
3. W buforze ConnectionConsumer znajdują się wybrane odwołania do komunikatów.
4. Opcja ConnectionConsumer żąda co najmniej jednej sesji ServerSessions z puli ServerSession.

5. ServerSessions (Sesje serwera) są przydzielane z puli ServerSession.
6. Obiekt ConnectionConsumer przypisuje odwołania do komunikatu do serwera ServerSessions i uruchamia wątki ServerSession działające.
7. Każda sesja ServerSession pobiera z niej przywoływane komunikaty. Przekazuje je do metody onMessage z obiektu MessageListener , który jest powiązany z sesją JMS .
8. Po zakończeniu przetwarzania wartość ServerSession jest zwracana do puli.

Serwer aplikacji zwykle dostarcza pulę ServerSessioni funkcję ServerSession .

z/OS Korzystanie z produktu IBM MQ classes for JMS na serwerze środowiska JVM środowiska OSGi produktu CICS

Obsługa przesyłania komunikatów oparta na standardach dla aplikacji działających w środowisku produktu CICS Open Services Gateway initiative (OSGi) Server jest udostępniana przy użyciu produktu IBM MQ classes for JMS.

Ostrzeżenie: Sprawdź wymagania systemowe dla systemu CICS używanego przez przedsiębiorstwo. Szczegółowe informacje na ten temat można znaleźć w sekcji Szczegółowe wymagania systemowe dla produktu CICS Transaction Server .

Z poziomu produktu IBM MQ 8.0produkt IBM MQ obsługuje korzystanie z produktu IBM MQ classes for JMS w określonych wersjach serwera CICS Open Services Gateway initiative (OSGi) Java Virtual Machine (JVM).

Ten temat stanowi wprowadzenie do konfigurowania produktu IBM MQ classes for JMS w środowisku serwera JVM.

Szczegółowe informacje na temat konfigurowania i konfigurowania systemu zawiera sekcja Korzystanie z produktu IBM MQ classes for JMS na serwerze JVM środowiska OSGi w dokumentacji produktu CICS .

Ograniczenia ogólne

W przypadku korzystania z produktu IBM MQ classes for JMS na serwerze CICS OSGi JVM obowiązują następujące ograniczenia:

- Połączenia w trybie klienta nie są obsługiwane.
- Połączenia są obsługiwane tylko dla menedżerów kolejek produktu IBM WebSphere MQ 7.1 lub IBM MQ 8.0 lub nowszych. Atrybut **PROVIDERVERSION** w fabryce połączeń musi być nieokreślony albo wartość większa niż lub równa siedmiu.
- Użycie dowolnego z fabryk połączeń XA, na przykład `com.ibm.mq.jms.MQXAConnectionFactory`, nie jest obsługiwane.
- Użycie produktu IBM MQ classes for JMS na serwerze CICS OSGi JVM jest obsługiwane tylko w produkcie CICS 5.2 lub nowszym. Jeśli używany jest produkt CICS 5.2, należy zastosować poprawkę APAR PI32151.

V 9.1.0 Przed IBM MQ 9.1.0 , jeśli użytkownik jest użytkownikiem produktu Long Term Support , lub przed IBM MQ 9.0.1 , jeśli użytkownik jest użytkownikiem produktu Continuous Delivery , użycie IBM MQ classes for JMS w środowisku serwera JVM produktu Liberty nie jest obsługiwane.

Zadania pokrewne

Konfigurowanie właściwości JMS **PROVIDERVERSION**

z/OS V 9.1.0 Korzystanie z produktu IBM MQ classes for JMS na serwerze JVM CICS Liberty

W programie IBM MQ 9.1.0programy Java działające na serwerze CICS Liberty JVM mogą korzystać z IBM MQ classes for JMS w celu uzyskania dostępu do produktu IBM MQ.

Musi być używana wersja IBM MQ 9.1.0 adaptera zasobów IBM MQ. Adapter zasobów można uzyskać z produktu Fix Central (patrz sekcja “Instalowanie adaptera zasobów w produkcie Liberty” na stronie 452).

Istnieją dwa rodzaje wirtualnych maszyn JVM profilu produktu Liberty dostępne w produkcie CICS 5.3 i nowszych wersjach, typy połączeń możliwe do IBM MQ są ograniczone w następujący sposób:

CICS Liberty Standard

- Adapter zasobów produktu IBM MQ może łączyć się z dowolną wersją usługi IBM MQ w trybie CLIENT.
- Adapter zasobów produktu IBM MQ może łączyć się z dowolną wersją usługi IBM MQ for z/OS w trybie BINDINGS, gdy nie ma połączenia CICS (aktywna definicja zasobu CICS MQCONN) z tym samym menedżerem kolejek z tego samego regionu CICS .

CICS Liberty Zintegrowane

- Adapter zasobów produktu IBM MQ może łączyć się z dowolną wersją usługi IBM MQ w trybie CLIENT.
- Połączenie trybu BINDINGS nie jest obsługiwane.

Szczegółowe informacje na temat konfigurowania i konfigurowania systemu zawiera sekcja Korzystanie z produktu IBM MQ classes for JMS na serwerze maszyny JVM serwera Liberty w dokumentacji produktu CICS .

Używanie produktu IBM MQ classes for JMS w produkcie IMS

Obsługa przesyłania komunikatów oparta na standardach w środowisku produktu IMS 13 jest udostępniana za pośrednictwem produktu IBM MQ classes for JMS.

Sprawdź wymagania systemowe dla systemu IMS używanego przez przedsiębiorstwo. Więcej informacji na ten temat zawiera sekcja Ogólne informacje dotyczące planowania dla systemu IMS 13 .

W produkcie IBM MQ 8.0.0 Fix Pack 4produkt IBM MQ obsługuje korzystanie z produktu IBM MQ classes for JMS w produkcie IMS w wersji 13 lub nowszej.

W tym zestawie tematów opisano sposób konfigurowania produktu IBM MQ classes for JMS w środowisku IMS oraz ograniczenia interfejsu API, które mają zastosowanie w przypadku korzystania z klasycznych (JMS 1.1) i uproszczonych (JMS 2.0) interfejsów. Sekcja “Ograniczenia funkcji API produktu JMS” na stronie 342 zawiera listę informacji specyficznych dla interfejsu API.

Uwaga: Podobne ograniczenia mają zastosowanie do wcześniejszych (JMS 1.0.2) interfejsów specyficznych dla domeny, ale nie są one opisane w szczególności w tym miejscu.

Obsługiwane regiony zależne IMS

Obsługiwane są następujące typy regionów zależnych:

- MPR
- BMP
- IFP
- JMP (tylko 31 bitowych maszyn wirtualnych Java (JVM), 64 bitowe maszyny JVM nie są obsługiwane)
- JBP (tylko 31-bitowe maszyny JVM, 64-bitowe maszyny JVM nie są obsługiwane)

O ile nie zostało to szczegółowo opisane w poniższych tematach, produkt IBM MQ classes for JMS zachowuje się tak samo we wszystkich typach regionów.

Obsługiwane maszyny wirtualne Java

IBM MQ classes for JMS requires Java Platform, Standard Edition 7 (Java SE 7) or later.

Inne ograniczenia

W przypadku korzystania z produktu IBM MQ classes for JMS w środowisku IMS obowiązują następujące ograniczenia:

- Połączenia w trybie klienta nie są obsługiwane.
- Połączenia są obsługiwane tylko dla menedżerów kolejek produktu IBM MQ 8.0 przy użyciu dostawcy przesyłania komunikatów IBM MQ `Normal`lub trybu `Version 8` .

  Atrybut **PROVIDERVERSION** w fabryce połączeń musi być nieokreślony albo wartość większa niż lub równa siedmiu.
- Użycie dowolnego z fabryk połączeń XA, na przykład `com.ibm.mq.jms.MQXAConnectionFactory`, nie jest obsługiwane.

Zadania pokrewne

Definiowanie IBM MQ do IMS

Konfigurowanie adaptera IMS do użytku z produktem IBM MQ classes for JMS

Produkt IBM MQ classes for JMS korzysta z tego samego adaptera IBM MQ-IMS , który jest używany przez inne języki programowania. Ten adapter używa zewnętrznego narzędzia dołączania podsystemu 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 produktu IMS , zgodnie z opisem w sekcji Konfigurowanie adaptera IMS.

Ostrzeżenie: Nie ma potrzeby wykonywania kroku opisowego, który opisuje budowanie dynamicznego kodu pośredniczącego, o ile nie jest potrzebny dynamiczny kod pośredniczący do innych celów.

Po skonfigurowaniu adaptera IMS należy wykonać poniższą procedurę.

Procedura

1. Zaktualizuj zmienną LIBPATH w elemencie systemu IMS PROCLIB, do którego odwołuje się parametr ENVIRON w zależnym regionie JCL (na przykład DFSJVMEV), tak aby obejmował on rodzime biblioteki produktu IBM MQ classes for JMS .

   Jest to katalog zFS , który zawiera `libmqjims.so`. Na przykład wartość DFSJVMEV może wyglądać tak, jak poniżej, gdzie ostatni wiersz to katalog zawierający rodzime biblioteki produktu IBM MQ classes for JMS :

```
LIBPATH=>
/java/java71_31/J7.1/bin/j9vm:>
/java/java71_31/J7.1/bin:>
/ims13/dbdc/imsjava/classic/lib:>
/ims13/dbdc/imsjava/lib:>
/mqm/V8R0M0/java/lib
```

2. Dodaj IBM MQ classes for JMS do ścieżki klas maszyny JVM używanej przez region zależny IMS , aktualizując opcję `java.class.path` .

   Wykonaj to zgodnie z instrukcjami w sekcji DFSJVMMS member of the IMS PROCLIB data set.

   Można na przykład użyć następującego miejsca, w którym wiersz pogrubiony oznacza aktualizację:

```
-Djava.class.path=/ims13/dbdc/imsjava/imsutm.jar:/ims13/dbdc/imsjava/imsudb.jar:
/mqm/V8R0M0/java/lib/com.ibm.mq.allclient.jarLIBPATH_SUFFIX=MQ_INSTALLATION_PATH
```

   Uwaga: Mimo że w katalogu zawierającym IBM MQ classes for JMSjest dostępnych wiele różnych plików jar, potrzebny jest tylko plik `com.ibm.mq.allclient.jar` .

3. Zatrzymaj i zrestartuj wszystkie zależne regiony produktu IMS , które będą korzystać z produktu IBM MQ classes for JMS.

Co dalej

Tworzenie i konfigurowanie fabryk połączeń i miejsc docelowych.

Istnieją trzy możliwe podejścia do 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 w aplikacji IBM MQ classes for JMS” na stronie 197.

Należy pamiętać, że te trzy podejścia są poprawne w środowisku IMS .

Zadania pokrewne

Konfigurowanie adaptera IMS

Definiowanie IBM MQ do IMS

Zachowanie transakcyjne

Komunikaty wysyłane i odbierane przez IBM MQ classes for JMS w środowisku IMS są zawsze powiązane z jednostką pracy IMS (UOW), która jest aktywna w bieżącym zadaniu.

Ta jednostka pracy może zostać zakończona tylko przez wywołanie metod zatwierdzania lub wycofania zmian w instancji obiektu `com.ibm.ims.dli.tm.Transaction` lub przez zadanie IMS kończące się normalnie, w którym to przypadku jednostka pracy jest niejawnie zatwierdzona. 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 kontekst sesji lub JMS jest tworzony przy użyciu jednego z następujących mechanizmów:

- `Connection.createSession(false, Session.AUTO_ACKNOWLEDGE)`
- `Connection.createSession(Session.AUTO_ACKNOWLEDGE)`
- `ConnectionFactory.createContext(JMSContext.AUTO_ACKNOWLEDGE)`

następnie zachowanie tej sesji lub kontekst JMS jest następujące:

- Wszystkie wysłane komunikaty są wysyłane poza UOW IMS . Oznacza to, że będą one dostępne w docelowym miejscu docelowym natychmiast lub gdy podany przedział czasu opóźnienia dostawy zostanie zakończony.
- Wszystkie nietrwałe komunikaty będą odbierane poza jednostką UOW IMS , chyba że właściwość SYNCPOINTALLGETS została określona w fabryce połączeń, która utworzyła kontekst sesji lub JMS .
- Komunikaty trwałe będą zawsze odbierane wewnątrz jednostki pracy produktu IMS .

Może to być przydatne, jeśli na przykład użytkownik chce napisać komunikat kontrolny do kolejki, nawet jeśli jednostka pracy wycofa się z kolejki.

Implikacje punktów synchronizacji produktu IMS

Produkt IBM MQ classes for JMS jest kompilowany przy użyciu istniejącej obsługi adaptera IBM MQ , która korzysta z funkcji ESAF. Oznacza to, że zastosowanie ma udokumentowane zachowanie, w tym wszystkie otwarte uchwyty, które są zamykane przez adapter IMS w momencie wystąpienia punktu synchronizacji.

Więcej informacji zawiera sekcja “Punkty synchronizacji w aplikacjach IMS” na stronie 71.

Aby zilustrować ten punkt, należy wziąć pod uwagę następujący kod działający w środowisku JMP. Drugie wywołanie funkcji `mp.send()` powoduje, że `JMSException` jako kod `messageQueue.getUnique(inputMessage)` powoduje zamknięcie wszystkich otwartych uchwytów obiektów i obiektów IBM MQ .

Podobne zachowanie jest obserwowane, jeśli wywołanie `getUnique()` zostało zastąpione `Transaction.commit()`, ale nie w przypadku użycia opcji `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 takim przypadku połączenie z produktem IBM MQ jest zamykane przed wywołaniem produktu `getUnique()`. Następnie połączenie i sesja są 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 korzystania z adaptera IMS

Należy zdawać sobie sprawę z następujących ograniczeń. Dla każdego menedżera kolejek może być używany tylko jeden uchwyt połączenia. W interakcji z produktem IBM MQ występują implikacje, gdy używany jest zarówno kod produktu JMS , jak i kod rodzimy. Istnieją ograniczenia dotyczące uwierzytelniania i autoryzacji połączenia.

Jeden uchwyt połączenia dla każdego menedżera kolejek

Tylko jeden uchwyt połączenia w danym momencie do konkretnego menedżera kolejek jest dozwolony w regionach zależnych IMS . Wszystkie kolejne próby nawiązania połączenia z tym samym menedżerem kolejek ponownie wykorzystują istniejący uchwyt.

To zachowanie nie powinno powodować żadnych problemów w aplikacji, która korzysta tylko z IBM MQ classes for JMS. Takie zachowanie może powodować problemy w aplikacjach, które wchodzą w interakcje z produktem IBM MQ, w przypadku używania zarówno produktu IBM MQ classes for JMS , jak i interfejsu MQI w języku rodzimym napisanych w językach, takich jak COBOL lub C.

Konsekwencje interakcji z produktem IBM MQ podczas używania zarówno produktu JMS , jak i kodu rodzimego.

Mogą wystąpić problemy podczas wychodzenia z kodu Java i kodu rodzimego, które korzystają z funkcji produktu IBM MQ , a także gdy połączenie z produktem IBM MQ nie jest zamknięte przed opuszczeniem kodu rodzimego lub kodu Java.

Na przykład w poniższym pseudo kodzie uchwyt połączenia z menedżerem kolejek jest początkowo ustanawiany w kodzie Java przy użyciu IBM MQ classes for JMS. Uchwyt połączenia jest ponownie wykorzystywany w kodzie COBOL i unieważniony w wyniku wywołania komendy MQDISC.

Następnym razem, gdy produkt IBM MQ classes for JMS będzie korzystać z 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 błąd MQRC_HCONN_ERROR.

Mimo że możliwe jest współużytkowanie uchwytów połączeń produktu IBM MQ między kodem rodzimym i kodem Java (na przykład poprzedni przykład może działać, jeśli nie było wywołania MQDISC) w ogóle, sprawdzoną procedurą jest zamknięcie wszystkich uchwytów połączeń przed zmianą kodu z kodu Java na kod rodzimy lub odwrotnie.

Uwierzytelnianie i autoryzacja połączenia

Specyfikacja JMS umożliwia określenie nazwy użytkownika i hasła na potrzeby uwierzytelniania i autoryzacji podczas tworzenia połączenia lub obiektu kontekstu produktu JMS .

Ta opcja nie jest obsługiwana w środowisku IMS . Próba utworzenia połączenia podczas określania nazwy użytkownika i hasła powoduje zgłoszenie wyjątku `JMS Exception` . Próba utworzenia kontekstu

produktu JMS podczas określania nazwy użytkownika i hasła powoduje, że JMSRuntimeException jest zgłaszany.

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 funkcji API produktu JMS

Z perspektywy specyfikacji produktu JMS produkt IBM MQ classes for JMS traktuje produkt IMS jako serwer aplikacji zgodny z produktem Java EE, który zawsze ma w toku transakcję JTA.

Na przykład nigdy nie można wywołać programu `javax.jms.Session.commit()` w systemie IMS, ponieważ specyfikacja JMS wskazuje, że nie można jej wywołać w komponencie EJB JEE lub kontenerze WWW, podczas gdy transakcja JTA jest w toku.

Wynika to z następujących ograniczeń dotyczących interfejsu API produktu JMS , oprócz tych opisanych w sekcji “Zachowanie transakcyjne” na stronie 339.

Ograniczenia dotyczące klasycznego interfejsu API

- `javax.jms.Connection.createConnectionConsumer(javax.jms.Destination, String, javax.jms.ServerSessionPool, int)` zawsze zgłasza JMSException.
- `javax.jms.Connection.createDurableConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int)` zawsze zgłasza JMSException.
- Wszystkie trzy warianty metody `javax.jms.Connection.createSession` zawsze zwracają wyjątek JMSException, jeśli dla połączenia już istnieje aktywna sesja.
- `javax.jms.Connection.createSharedConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int)` zawsze zgłasza JMSException.
- `javax.jms.Connection.createSharedDurableConnectionConsumer(javax.jms.Topic, String, String, javax.jms.ServerSessionPool, int)` zawsze zgłasza JMSException.
- `javax.jms.Connection.setClientID()` zawsze zgłasza JMSException.
- `javax.jms.Connection.setExceptionListener(javax.jms.ExceptionListener)` zawsze zgłasza JMSException.
- `javax.jms.Connection.stop()` zawsze zgłasza JMSException.
- `javax.jms.MessageConsumer.setMessageListener(javax.jms.MessageListener)` zawsze zgłasza JMSException.
- `javax.jms.MessageConsumer.getMessageListener()` zawsze zgłasza JMSException.
- `javax.jms.MessageProducer.send(javax.jms.Destination, javax.jms.Message,javax.jms.CompletionListener)` zawsze zgłasza JMSException.
- `javax.jms.MessageProducer.send(javax.jms.Destination, javax.jms.Message, int, int, long, javax.jms.CompletionListener)` zawsze zgłasza JMSException.
- `javax.jms.MessageProducer.send(javax.jms.Message, int, int, long, javax.jms.CompletionListener)` zawsze zgłasza JMSException.
- `javax.jms.MessageProducer.send(javax.jms.Message,javax.jms.CompletionListener)` zawsze zgłasza JMSException.
- `javax.jms.Session.run()` zawsze zgłasza JMSRuntimeException.
- `javax.jms.Session.setMessageListener(javax.jms.MessageListener)` zawsze zgłasza JMSException.
- `javax.jms.Session.getMessageListener()` zawsze zgłasza JMSException.

Uproszczone ograniczenia interfejsu API

- `javax.jms.JMSContext.createContext(int)` zawsze zgłasza `JMSRuntimeException`.
- `javax.jms.JMSContext.setClientID(String)` zawsze zgłasza `JMSRuntimeException`.
- `javax.jms.JMSContext.setExceptionListener(javax.jms.ExceptionListener)` zawsze zgłasza `JMSRuntimeException`.
- `javax.jms.JMSContext.stop()` zawsze zgłasza `JMSRuntimeException`.
- `javax.jms.JMSProducer.setAsync(javax.jms.CompletionListener)` zawsze zgłasza `JMSRuntimeException`.

użycieIBM MQ classes for Java

Użyj produktu IBM MQ w środowisku Java . Program IBM MQ classes for Java umożliwia aplikacji Java łączenie się z serwerem IBM MQ jako klient IBM MQ lub bezpośrednie połączenie z menedżerem kolejek produktu IBM MQ .

Uwaga:

Produkt IBM nie rozszerzy IBM MQ classes for Java i nie będzie on funkcjonalnie stabilizowany na poziomie dostarczonym w produkcie IBM MQ 8.0. Istniejące aplikacje, które korzystają z produktu IBM MQ classes for Java , nadal są w pełni obsługiwane, ale nowe funkcje nie zostaną dodane, a żądania dotyczące rozszerzeń zostaną odrzucone. W pełni obsługiwane oznacza, że defekty zostaną naprawione wraz ze zmianami wymaganowymi przez zmiany w wymaganiach systemowych produktu IBM MQ .

IBM MQ classes for Java nie są obsługiwane w produkcie IMS.

IBM MQ classes for Java nie są obsługiwane w produkcie WebSphere Liberty. Nie mogą one być używane z funkcją przesyłania komunikatów IBM MQ Liberty ani z ogólną obsługą produktu JCA . Więcej informacji na ten temat zawiera sekcja Korzystanie z interfejsów Java produktu WebSphere MQ w środowiskach J2EE/JEE.

Produkt IBM MQ classes for Java jest jednym z dwóch alternatywnych interfejsów API, które mogą być używane przez aplikacje produktu Java w celu uzyskania dostępu do zasobów produktu IBM MQ . Innym interfejsem API jest IBM MQ classes for JMS.

W produkcie IBM MQ 8.0produkt IBM MQ classes for Java jest budowany z produktem Java 7.

Środowisko wykonawcze produktu Java 7 obsługuje uruchamianie wcześniejszych wersji plików klas.

IBM MQ classes for Java obudowuje interfejs kolejki komunikatów (Message Queue Interface-MQI), rodzimy interfejs API produktu IBM MQ i korzysta z podobnego modelu obiektowego w interfejsach języka C++ i .NET do produktu IBM MQ.

Opcje programowalne umożliwiają IBM MQ classes for Java łączenie się z programem IBM MQ w jeden z następujących sposobów:

- W trybie klienta jako IBM MQ MQI client przy użyciu protokołu TCP/IP (Transmission Control Protocol/ Internet Protocol (TCP/IP))
- W trybie powiązańłączy się bezpośrednio z serwerem IBM MQ przy użyciu interfejsu JNI (Java Native Interface).

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 łączyć się z dowolnym obsługiwanym menedżerem kolejek przy użyciu trybu klienta.

Aby nawiązać połączenie z menedżerem kolejek w trybie klienta, aplikacja IBM MQ classes for Java może działać w tym samym systemie, w którym jest uruchomiony 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 sposobu pisania aplikacji w celu korzystania z połączeń w trybie klienta zawiera sekcja “Tryby połączenia IBM MQ classes for Java” na stronie 368.

połączenie w trybie powiązań

W przypadku użycia w trybie powiązań produkt IBM MQ classes for Java używa interfejsu JNI (Java Native Interface) do wywołania bezpośrednio do istniejącego interfejsu API menedżera kolejek, zamiast komunikowania się przez sieć. W większości środowisk łączenie w trybie powiązań zapewnia lepszą wydajność aplikacji IBM MQ classes for Java niż łączenie w trybie klienta, dzięki uniknięciu kosztów komunikacji TCP/IP.

Aplikacje, które używają programu IBM MQ classes for Java do łączenia w trybie powiązań, muszą działać w tym samym systemie, co menedżer kolejek, z którym się łączy.

Środowisko wykonawcze programu Java , które jest używane do uruchamiania aplikacji IBM MQ classes for Java , musi być skonfigurowane pod kątem ładowania bibliotek produktu IBM MQ classes for Java . Aby uzyskać więcej informacji, należy zapoznać się z informacjami w sekcji “IBM MQ classes for Java biblioteki” na stronie 353 .

Więcej informacji na temat sposobu pisania aplikacji w celu korzystania z połączeń w trybie powiązań zawiera sekcja “Tryby połączenia IBM MQ classes for Java” na stronie 368.

Pojęcia pokrewne

Interfejsy językowe programu IBM MQ Java

“użycieIBM MQ classes for JMS” na stronie 83

IBM MQ classes for Java Message Service (IBM MQ classes for JMS) jest dostawcą JMS dostarczonym z produktem IBM MQ. Oprócz implementowania interfejsów zdefiniowanych w pakiecie javax.jms produkt IBM MQ classes for JMS udostępnia dwa zestawy rozszerzeń do interfejsu API produktu JMS .

Zadania pokrewne

Śledzenie aplikacji IBM MQ classes for Java

Rozwiązywanie problemów z Java i JMS

Dlaczego należy używać produktu IBM MQ classes for Java?

A Java application can use either IBM MQ classes for Java or IBM MQ classes for JMS to access IBM MQ resources.

Uwaga: Mimo że 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 JMS. Funkcje, które ostatnio zostały dodane do produktu IBM MQ, takie jak asynchroniczne zużywanie i automatyczne ponowne połączenie, nie są dostępne w IBM MQ classes for Java, ale są dostępne w IBM MQ classes for JMS. Więcej informacji na ten temat zawiera sekcja “Dlaczego należy używać produktu IBM MQ classes for JMS?” na stronie 84.

Uwaga: Produkt IBM MQ classes for Java jest stabilny pod względem funkcjonalnym na poziomie dostarczonym w produkcie IBM MQ 8.0. Istniejące aplikacje, które korzystają z produktu IBM MQ classes for Java , będą nadal w pełni obsługiwane, ale ta funkcja API jest stabilna, dlatego nowe funkcje nie zostaną dodane, a żądania dotyczące udoskonaleń zostały odrzucone. W pełni obsługiwane oznacza, że defekty zostaną naprawione wraz ze zmianami wymaganowymi przez zmiany w wymaganiach systemowych produktu IBM MQ .

Wymagania wstępne dla produktu IBM MQ classes for Java

Aby używać produktu IBM MQ classes for Java, potrzebne są inne produkty oprogramowania.

Informacje na temat wymagań wstępnych dla produktu IBM MQ classes for Javamożna znaleźć na stronie WWW produktu Wymagania systemowe produktu IBM MQ .

Aby tworzyć aplikacje produktu IBM MQ classes for Java , potrzebny jest pakiet Java Development Kit (JDK). Szczegółowe informacje na temat pakietów JDK obsługiwanych przez system operacyjny można znaleźć w informacjach o produkcie Wymagania systemowe produktu IBM MQ .

Aby uruchomić aplikacje produktu IBM MQ classes for Java , potrzebne są następujące komponenty oprogramowania:

- Menedżer kolejek produktu IBM MQ dla aplikacji, które łączą się z menedżerem kolejek.
- Środowisko wykonawcze programów Java (JRE) dla każdego systemu, w którym uruchamiane są aplikacje. Odpowiednie środowisko JRE jest dostarczane razem z produktem IBM MQ.
- 

Dla IBM i, QShell, która jest opcją 30 systemu operacyjnego
- 

Dla z/OS, UNIX and Linux System Services (USS)

Jeśli do korzystania z modułów szyfrujących zgodnych ze standardem FIPS 140-2 wymagane są połączenia TLS, wymagany jest dostawca FIPS IBM Java JSSE (IBMJSSEFIPS). Każdy pakiet IBM JDK i JRE w wersji 1.4.2 lub nowszej zawiera IBMJSSEFIPS.

W aplikacjach IBM MQ classes for Java można używać adresów Internet Protocol wersja 6 (IPv6), jeśli produkt IPv6 jest obsługiwany przez wirtualną maszynę Java (JVM) i implementację protokołu TCP/IP w systemie operacyjnym.

Uruchamianie aplikacji produktu IBM MQ classes for Java w produkcie Java EE

Przed użyciem produktu IBM MQ classes for Java w produkcie Java EEnależy wziąć pod uwagę pewne ograniczenia i uwagi dotyczące projektowania.

Produkt IBM MQ classes for Java ma ograniczenia w przypadku użycia 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 , która działa w środowisku produktu Java EE . Te ograniczenia i uwagi są 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 . Mimo że aplikacja w ramach elementu sterującego JTA może korzystać z produktu IBM MQ classes for Java, każda praca wykonana przez te klasy nie jest kontrolowana przez jednostki pracy JTA. Zamiast tego tworzą one lokalne jednostki pracy oddzielone od tych zarządzanych przez serwer aplikacji za pośrednictwem interfejsów JTA. W szczególności wycofanie zmian w transakcji JTA nie powoduje wycofania żadnych wysłanych ani odebranych komunikatów. To ograniczenie dotyczy transakcji zarządzanych aplikacji lub komponentów bean oraz do kontenerów zarządzanych przez kontener i wszystkich kontenerów produktu Java EE . Aby można było wykonywać operacje przesyłania komunikatów bezpośrednio z produktem IBM MQ w ramach transakcji koordynowanych przez serwer aplikacji, należy zamiast niego użyć produktu IBM MQ classes for JMS .

Tworzenie wątków

Produkt IBM MQ classes for Java tworzy wątki wewnętrznie dla różnych operacji. Na przykład podczas uruchamiania w trybie BINDINGS w celu bezpośredniego połączenia z lokalnym menedżerem kolejek wywołania są wykonywane w wątku "worker" utworzonym wewnętrznie przez produkt 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/subskrypcji.

Niektóre aplikacje produktu Java EE (na przykład te działające w kontenerach EJB i WWW) nie mogą tworzyć nowych wątków. Zamiast tego wszystkie prace muszą być wykonywane na głównych wątkach aplikacji zarządzanych przez serwer aplikacji. Gdy 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 wcześniej opisane wątki powodują, że aplikacja nie jest zgodna ze specyfikacją kontenera.

Produkt IBM MQ classes for JMS nie przerywa tych specyfikacji produktu Java EE i dlatego może być używany.

Ograniczenia dotyczące bezpieczeństwa

Strategie bezpieczeństwa implementowane przez serwer aplikacji mogą uniemożliwić wykonywanie niektórych operacji wykonywanych przez interfejs API produktu IBM MQ classes for Java , takich jak tworzenie i wykonywanie nowych wątków sterowania (zgodnie z opisem podanym w poprzednich sekcjach).

Na przykład serwery aplikacji zwykle działają z wyłączonymi zabezpieczeniami produktu Java i umożliwiają jej włączenie za pomocą konkretnej konfiguracji specyficznej dla serwera aplikacji (niektóre serwery aplikacji zezwalają również na bardziej szczegółową konfigurację strategii używanych w zabezpieczeniach produktu Java). Jeśli zabezpieczenia produktu Java są włączone, produkt IBM MQ classes for Java może złamać reguły wielowątkowości strategii bezpieczeństwa produktu Java zdefiniowane dla serwera aplikacji, a interfejs API może nie być w stanie utworzyć wszystkich wątków, których potrzebuje w celu funkcjonowania. Aby zapobiec problemom z zarządzaniem wątkami, korzystanie z produktu IBM MQ classes for Java nie jest obsługiwane w środowiskach, w których zabezpieczenia produktu Java są włączone.

Uwagi dotyczące odseparowania aplikacji

Korzyścią dla uruchamiania aplikacji w środowisku produktu Java EE jest izolacja aplikacji. Projektowanie i implementacja produktu IBM MQ classes for Java data predate środowiska Java EE . Produkt IBM MQ classes for Java może być używany w sposób, który nie obsługuje pojęcia izolacji aplikacji. Do konkretnych przykładów rozważań w tym zakresie należą:

- Użycie ustawień statycznych (całego procesu maszyny JVM) w ramach klasy MQEnvironment, takich jak:
  - ID użytkownika i hasło, które mają być używane do identyfikacji i uwierzytelniania połączenia
  - nazwa hosta, port i kanał używany dla połączeń klienckich
  - Konfiguracja TLS dla bezpiecznych połączeń klientów

  Zmodyfikowanie wszystkich właściwości MQEnvironment na korzyść jednej aplikacji ma również wpływ na inne aplikacje korzystające z tych samych właściwości. W przypadku uruchamiania w środowisku z wieloma aplikacjami, takimi jak Java EE, każda aplikacja musi używać własnej odrębnej konfiguracji poprzez utworzenie obiektów MQQueueManager z określonym zestawem właściwości, a nie domyślnych dla właściwości skonfigurowanych w klasie MQEnvironment w całej procesie.
- Klasa MQEnvironment wprowadza wiele metod statycznych, które działają globalnie dla wszystkich aplikacji korzystających z produktu 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 pliku kluczy
  - konfigurowanie wyjść kanału klienta
  - włączanie lub wyłączanie śledzenia diagnostycznego
  - zarządzanie domyślną pulą połączeń używaną do optymalizacji korzystania z połączeń z menedżerami kolejek

  Wywoływanie takich metod ma wpływ na wszystkie aplikacje działające w tym samym środowisku produktu Java EE .
- Zestawianie połączeń jest włączone w celu zoptymalizowania procesu nawiązywania wielu połączeń z tym samym menedżerem kolejek. Domyślny menedżer puli połączeń jest przetwarzalny w całym procesie i współużytkowany przez wiele aplikacji. Zmiany w konfiguracji puli połączeń, takie jak zastąpienie domyślnego menedżera połączeń dla jednej aplikacji za pomocą metody MQEnvironment.setDefaultConnectionManager(), wpływają w związku z tym na inne aplikacje działające na tym samym serwerze aplikacji Java EE.

- Protokół TLS jest konfigurowany dla aplikacji korzystających z produktu IBM MQ classes for Java przy użyciu klasy MQEnvironment i właściwości obiektu MQQueueManager . Nie jest ona zintegrowana z konfiguracją zabezpieczeń zarządzanych samego serwera aplikacji. Należy upewnić się, że produkt IBM MQ classes for Java został skonfigurowany w odpowiedni sposób, aby zapewnić wymagany poziom zabezpieczeń, a także nie korzystać z konfiguracji serwera aplikacji.

Ograniczenia trybu powiązań

Produkty IBM MQ i WebSphere Application Server mogą być zainstalowane na tym samym komputerze, na przykład, że główne wersje menedżera kolejek i adaptera zasobów produktu IBM MQ (RA) dostarczane w produkcie WebSphere Application Server są różne. Na przykład WebSphere Application Server 7.0, który jest dostarczany na poziomie IBM MQ RA 7.0.1, może być zainstalowany na tym samym komputerze, co menedżer kolejek produktu IBM WebSphere MQ 6 .

Jeśli główne wersje menedżera kolejek i adaptera zasobów są różne, nie można używać połączeń powiązań. Wszystkie połączenia z programu WebSphere Application Server do menedżera kolejek przy użyciu adaptera zasobów muszą używać połączeń typu klienta. Połączenia powiązań mogą być używane, jeśli wersje są takie same.

Konwersje łańcucha znaków w programie IBM MQ classes for Java

Opcje CharsetEncoders i CharsetDecoders produktu IBM MQ classes for Java są używane bezpośrednio do konwersji łańcuchów znaków. Domyślne zachowanie konwersji łańcucha znaków można skonfigurować przy użyciu dwóch właściwości systemowych. Obsługa komunikatów zawierających znaki niemożliwe do odwzorowania może być skonfigurowana za pomocą programu `com.ibm.mq.MQMD`.

Przed programem IBM MQ 8.0konwersje łańcuchów w produkcie IBM MQ classes for Java zostały wykonane przez wywołanie metod `java.nio.charset.Charset.decode(ByteBuffer)` i `Charset.encode(CharBuffer)` .

Użycie jednej z tych metod spowoduje, że zostanie użyta wartość domyślna (REPLACE) zniekształcona lub nieprzetłumaczalne dane. To zachowanie może spowodować błędy w aplikacjach i prowadzić do wystąpienia nieoczekiwanych znaków, na przykład ?, w przetłumaczonych danych.

Aby produkt IBM MQ 8.0mógł wcześniej i skuteczniej wykrywać takie problemy, należy IBM MQ classes for Java użyć komendy CharsetEncoders i CharsetDecoders bezpośrednio i jawnie skonfigurować obsługę zniekształcanych i nieprzetłumaczalnych danych. Domyślnym zachowaniem jest REPORT takie problemy, zgłaszając odpowiedni `MQException`.

Konfigurowanie

Translacja z UTF-16 (reprezentacja znakowa używana w produkcie Java) do rodzimego zestawu znaków, na przykład UTF-8, jest termed `encoding`, podczas gdy translacja w przeciwnym kierunku jest termed `decoding`.

Obecnie dekodowanie odbywa się domyślnie w przypadku produktu `CharsetDecoders`, zgłaszając błędy, zgłaszając wyjątek.

Jedno ustawienie służy do określenia `java.nio.charset.CodingErrorAction` , aby sterować obsługą błędów przy kodowaniu i dekodowaniu. Jedno z pozostałych ustawień jest używane do sterowania bajtem zastępczym lub bajtami, gdy kodowanie jest używane. Domyślny łańcuch zastępujący Java będzie używany w operacjach dekodowania.

Konfiguracja obsługi danych niepodlegających tłumaczom w produkcie IBM MQ classes for Java

W produkcie IBM MQ 8.0produkt `com.ibm.mq.MQMD` zawiera następujące dwa pola:

byte [] unMappablezamiennik
Sekwencja bajtów, która zostanie zapisana w zakodowanym łańcuchu, jeśli znak wejściowy nie może zostać przetłumaczony, a użytkownik określił REPLACE.

Wartość domyślna: "?".getBytes()
Domyślny łańcuch zastępujący Java jest używany w operacjach dekodowania.

java.nio.charset.CodingErrorAction unmappableAction
Określa działanie, które ma zostać podjęte w przypadku nieprzetłumaczalnych danych na temat kodowania i dekodowania:

Wartość domyślna: CodingErrorAction.REPORT;

Właściwości systemowe służące do ustawiania wartości domyślnych systemu

W produkcie IBM MQ 8.0dostępne są następujące dwie właściwości systemowe produktu Java w celu skonfigurowania domyślnego zachowania w odniesieniu do konwersji łańcuchów znaków.

com.ibm.mq.cfg.jmqi.UnmappableCharacterAction
Określa działanie, które ma zostać podjęte w przypadku nieprzetłumaczalnych danych na temat kodowania i dekodowania. Wartością może być REPORT, REPLACElub IGNORE.

com.ibm.mq.cfg.jmqi.UnmappableCharacterReplacement
Ustawia lub pobiera bajty zastępcze, które mają być stosowane, gdy znak nie może być odwzorowany w operacji kodowania. Domyślny łańcuch zastępujący Java jest używany podczas dekodowania operacji.

Aby uniknąć nieporozumień między znakami Java a reprezentacjami bajtowymi, należy podać `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, podczas gdy jest 111, jeśli rodzimy zestaw znaków jest ustawiony na EBCDIC.

Uwaga: Należy zwrócić uwagę, że jeśli obiekt MQMD lub MQMessage ma ustawione pola **`unmappableAction`** lub **`unMappableReplacement`** , wówczas wartości tych pól mają pierwszeństwo przed właściwościami systemu Java . Pozwala to na nadpisanie wartości określonych przez właściwości systemowe Java dla każdego komunikatu, jeśli jest to wymagane.

Pojęcia pokrewne
“Konwersje łańcucha znaków w programie IBM MQ classes for JMS” na stronie 133
Opcje CharsetEncoders i CharsetDecoders produktu IBM MQ classes for JMS są używane bezpośrednio do konwersji łańcuchów znaków. Domyślne zachowanie konwersji łańcucha znaków można skonfigurować przy użyciu dwóch właściwości systemowych. Obsługa komunikatów, które zawierają znaki bez odwzorowania, można skonfigurować za pomocą właściwości komunikatu w celu ustawienia działania UnmappableCharacteri bajtów zastępczych.

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, a także opisuje sposób konfigurowania produktu IBM MQ classes for Java po instalacji.

Co jest zainstalowane w systemie IBM MQ classes for Java

Najnowsza wersja produktu IBM MQ classes for Java jest instalowana wraz z produktem IBM MQ. Może być konieczne przesłonięcie domyślnych opcji instalacji w celu upewniania się, że jest to wykonane.

Więcej informacji na temat instalowania produktu IBM MQ zawiera sekcja:

- Multi Instalowanie oprogramowania IBM MQ
- z/OS Instalowanie produktu IBM MQ for z/OS

IBM MQ classes for Java są zawarte w plikach archiwum Java (JAR), `com.ibm.mq.jari com.ibm.mq.jmqi.jar`.

Obsługa standardowych nagłówków komunikatów, takich jak programowalny format komend (Programmable Command Format-PCF), znajduje się w pliku JAR `com.ibm.mq.headers.jar`.

Obsługa formatu PCF (Programmable Command Format-PCF) jest zawarta w pliku JAR `com.ibm.mq.pcf.jar`.

Uwaga: Nie zaleca się używania serwera IBM MQ classes for Java w obrębie serwera aplikacji. Więcej informacji na temat ograniczeń, które mają zastosowanie podczas pracy w tym środowisku, zawiera sekcja "Uruchamianie aplikacji produktu IBM MQ classes for Java w produkcie Java EE" na stronie 345. Więcej informacji na ten temat zawiera sekcja Korzystanie z interfejsów Java produktu WebSphere MQ w środowiskach J2EE/JEE.

Ważne: Oprócz przemieszczalnych plików JAR opisanych w "IBM MQ classes for Java relokacyjne pliki JAR" na stronie 349, kopiowanie plików JAR IBM MQ classes for Java lub rodzimych bibliotek do innych komputerów lub do innego położenia na komputerze, na którym zainstalowano produkt IBM MQ classes for Java , nie jest obsługiwane.

IBM MQ classes for Java relokacyjne pliki JAR
Możliwe do ponownego przydzielenia pliki JAR można przenieść do systemów, które muszą uruchamiać program IBM MQ classes for Java.

Ważne:

- Oprócz przemieszczalnych plików JAR opisanych w sekcji Pliki JAR do przemieszczenia, kopiowanie plików JAR IBM MQ classes for Java lub rodzimych bibliotek do innych komputerów lub do innego miejsca na komputerze, na którym zainstalowano produkt IBM MQ classes for Java , nie jest obsługiwane.
- Aby uniknąć konfliktów programu ładującego klasy, nie zaleca się pakowaniu przemieszczalnych plików JAR w obrębie wielu aplikacji w tym samym środowisku wykonawczym produktu Java . W tym scenariuszu należy rozważyć udostępnienie przemieszczalnych plików JAR produktu IBM MQ w ścieżce klasy środowiska wykonawczego produktu Java .
- Nie należy uwzględniać relokacyjnych plików JAR w aplikacjach wdrożonych na serwerach aplikacji Java EE , takich jak WebSphere Application Server. W tych środowiskach adapter zasobów produktu IBM MQ powinien zostać wdrożony i używany zamiast tego, ponieważ zawiera on IBM MQ classes for Java. Należy zauważyć, że produkt WebSphere Application Server osadza adapter zasobów produktu IBM MQ , dlatego nie ma potrzeby ręcznego wdrażania tego adaptera 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 448.
- W przypadku tworzenia pakunków przemieszczalnych plików JAR w aplikacjach, należy uwzględnić wszystkie wstępnie wymagane pliki JAR zgodnie z opisem w sekcji Pliki JAR z produktem Relocatable JAR. Należy również upewnić się, że istnieją odpowiednie procedury aktualizacji dołączonych plików JAR w ramach obsługi aplikacji, aby upewnić się, że IBM MQ classes for Java pozostają aktualne, a znane problemy są ponownie mediowane.

Przemieszczalne pliki JAR

W obrębie przedsiębiorstwa następujące pliki mogą być przenoszone do systemów, które muszą uruchamiać aplikacje produktu IBM MQ classes for Java :

- `com.ibm.mq.allclient.jar`
- `com.ibm.mq.traceControl.jar`
- `bcpkix-jdk15on.jar`
- `bcprov-jdk15on.jar`
- V 9.1.0.9 `bcutil-jdk15on.jar`
- V 9.1.2 `org.json.jar`

Wymagane są pliki JAR obsługi zabezpieczeń Zamku Bouncy oraz pliki JAR obsługi CMS. Więcej informacji na ten temat zawiera sekcja Wsparcie dla środowisk JRE innych niżIBM z AMS. Wymagane są następujące pliki JAR:

- `bcpkix-jdk15on.jar`

- bcprov-jdk15on.jar
- V 9.1.0.9 bcutil-jdk15on.jar

V 9.1.2 Plik org.json.jar jest wymagany, jeśli aplikacja IBM MQ classes for Java korzysta z tabeli definicji kanału klienta w formacie JSON.

Plik com.ibm.mq.allclient.jar zawiera IBM MQ classes for JMS, IBM MQ classes for Javaoraz klasy PCF i Headers. Jeśli ten plik zostanie przeniesiony do nowego położenia, należy się upewnić, że nowe położenie zostało zachowane przy użyciu nowych pakietów poprawek produktu IBM MQ , które zostały zachowane. Należy również upewnić się, że użycie tego pliku jest znane z działu wsparcia produktu IBM , jeśli jest otrzymana poprawka tymczasowa.

Aby określić wersję pliku com.ibm.mq.allclient.jar , należy użyć następującej komendy:

```
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:    9.1.0.0
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:    9.1.0.0
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:    9.1.0.0
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:    9.1.0.0
Level:      p000-L140428.1
Build Type: Production
Location:   file:/C:/Program Files/IBM/MQ_1/java/lib/com.ibm.mq.allclient.jar
```

Plik com.ibm.mq.traceControl.jar jest używany do dynamicznego sterowania śledzeniem dla aplikacji IBM MQ classes for JMS . Więcej informacji na ten temat zawiera sekcja Kontrolowanie śledzenia w działającym procesie przy użyciu klas produktu IBM MQ dla języka Java i klas produktu IBM MQ dla usługi JMS.

Katalogi instalacyjne produktu IBM MQ classes for Java
Pliki i przykłady produktu IBM MQ classes for Java są instalowane w różnych miejscach, w zależności od platformy. Położenie środowiska wykonawczego produktu Java (JRE), które jest instalowane razem z produktem IBM MQ , jest również różne w zależności od platformy.

Katalogi instalacyjne dla plików IBM MQ classes for Java

Tabela 48 na stronie 350 informuje, gdzie są zainstalowane pliki IBM MQ classes for Java .

Tabela 48. IBM MQ classes for Java Katalogi instalacyjne

Platforma	**Katalog**
AIX AIX	*MQ_INSTALLATION_PATH*/java/lib
	/QIBM/ProdData/mqm/java/lib

Tabela 48. IBM MQ classes for Java Katalogi instalacyjne (kontynuacja)	
Platforma	**Katalog**
Linux	*MQ_INSTALLATION_PATH*/java/lib
Solaris	*MQ_INSTALLATION_PATH*/java/lib
Windows	*MQ_INSTALLATION_PATH*\java\lib
z/OS	*MQ_INSTALLATION_PATH*/mqm/V8R0M0/java /lib

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Katalogi instalacyjne dla przykładów

Niektóre aplikacje przykładowe, takie jak programy weryfikacji instalacji (Installation Verification Programs-IVPs), są dostarczane razem z produktem IBM MQ. Tabela 49 na stronie 351 wskazuje miejsce, w którym zainstalowane są przykładowe aplikacje. Przykłady IBM MQ classes for Java znajdują się w podkatalogu o nazwie `wmqjava`. Przykłady PCF znajdują się w podkatalogu o nazwie `pcf`.

Tabela 49. Katalogi przykładów	
Platforma	**Katalog**
AIX	*MQ_INSTALLATION_PATH*/samp/wmqjava/
IBM i	/QIBM/ProdData/mqm/java/samples
Linux	*MQ_INSTALLATION_PATH*/samp/wmqjava/
Solaris	*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 najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Katalogi instalacyjne dla środowiska JRE

IBM MQ classes for JMS wymaga Java 7 (lub wyższej) środowiska wykonawczego Java (JRE). Odpowiednie środowisko JRE jest instalowane razem z produktem IBM MQ. Tabela 50 na stronie 351 informuje, gdzie jest zainstalowane środowisko JRE. Aby uruchomić program Java , taki jak dostarczone przykłady, używając tego środowiska JRE, jawnie wywołaj produkt *JRE_LOCATION*`/bin/java` lub dodaj produkt *JRE_LOCATION*`/bin` do środowiska PATH (lub równoważnego) dla używanej platformy, gdzie *JRE_LOCATION* jest katalogiem podanym w produkcie Tabela 50 na stronie 351.

Tabela 50. Katalogi środowiska JRE	
Platforma	**Katalog**
AIX	*MQ_INSTALLATION_PATH*/java/jre
IBM i	/QIBM/ProdData/mqm/java/jre

Tabela 50. Katalogi środowiska JRE (kontynuacja)	
Platforma	**Katalog**
Linux	*MQ_INSTALLATION_PATH*/java/jre
Solaris	*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 najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Zmienne środowiskowe istotne dla IBM MQ classes for Java
Aby uruchomić aplikacje produktu IBM MQ classes for Java , ich ścieżka klas musi zawierać katalogi IBM MQ classes for Java i przykładowe katalogi.

W przypadku produktu IBM MQ classes for Java, aplikacje do uruchomienia, ścieżka klasy musi zawierać odpowiedni katalog IBM MQ classes for Java . Aby uruchomić przykładowe aplikacje, ścieżka klasy musi także zawierać odpowiednie katalogi przykładów. Te informacje można podać w komendzie wywołania Java lub w zmiennej środowiskowej CLASSPATH.

Ważne: Ustawienie opcji Java *-Xbootclasspath*, tak aby obejmował IBM MQ classes for Java, nie jest obsługiwane.

W produkcie Tabela 51 na stronie 352 jest wyświetlane odpowiednie ustawienie CLASSPATH, które ma być używane na każdej platformie do uruchamiania aplikacji produktu IBM MQ classes for Java , w tym przykładowych aplikacji.

Tabela 51. Ustawienie CLASSPATH w celu uruchamiania aplikacji produktu IBM MQ classes for Java , w tym przykładowych aplikacji produktu IBM MQ classes for Java .	
Platforma	**Ustawienie CLASSPATH**
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:
Solaris	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/V9R1M0/java/lib/com.ibm.mq.jar: *MQ_INSTALLATION_PATH*/mqm/V9R1M0/java/samples/wmqjava: *MQ_INSTALLATION_PATH*/mqm/V9R1M0/java/samples/pcf

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

W przypadku kompilacji za pomocą opcji -Xlint może zostać wyświetlony komunikat z ostrzeżeniem, że plik com.ibm.mq.ese.jar nie jest obecny. Ostrzeżenie można zignorować. Ten plik jest dostępny tylko wtedy, gdy zainstalowany jest produkt Advanced Message Security.

Skrypty dostarczane z produktem IBM MQ classes for JMS korzystają z następujących zmiennych środowiskowych:

MQ_JAVA_DATA_PATH
Ta zmienna środowiskowa określa katalog dla danych wyjściowych dziennika i śledzenia.

MQ_JAVA_INSTALL_PATH
Ta zmienna środowiskowa określa katalog, w którym zainstalowano produkt IBM MQ classes for Java , jak to pokazano w sekcji Katalogi instalacyjne produktu IBM MQ classes for Java.

MQ_JAVA_LIB_PATH
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 dostarczane z produktem IBM MQ classes for Java, takie jak IVTRun, używają tej zmiennej środowiskowej.

Windows W systemie Windowswszystkie zmienne środowiskowe są ustawiane automatycznie podczas instalacji.

UNIX W systemie UNIXmożna użyć skryptu **setjmsenv** (jeśli używana jest 32-bitowa maszyna JVM) lub **setjmsenv64** (jeśli używana jest 64-bitowa maszyna JVM) do ustawienia zmiennych środowiskowych.

Linux UNIX W systemie UNIX and Linuxte skrypty znajdują się w katalogu *MQ_INSTALLATION_PATH*/java/bin .

IBM i W systemie IBM izmienna środowiskowa QIBM_MULTI_THREADED musi być ustawiona na wartość Y. Aplikacje wielowątkowe można uruchamiać w ten sam sposób, w jaki uruchamiane są aplikacje jednowątkowe. Więcej informacji na ten temat zawiera sekcja Konfigurowanie produktu IBM MQ z językiem Java i JMS .

Produkt IBM MQ classes for Java wymaga środowiska wykonawczego Java 7 Java (JRE). Informacje na temat położenia odpowiedniego środowiska JRE zainstalowanego razem z produktem IBM MQmożna znaleźć w sekcji “Katalogi instalacyjne produktu IBM MQ classes for Java” na stronie 350.

IBM MQ classes for Java biblioteki
Położenie bibliotek produktu IBM MQ classes for Java jest różne w zależności od platformy. Tę lokalizację należy określić podczas uruchamiania aplikacji.

Aby określić położenie bibliotek JNI (Native Interface) produktu Java , należy uruchomić aplikację za pomocą komendy **java** z następującym formatem:

```
java -Djava.library.path= library_path application_name
```

gdzie *ścieżka_biblioteki_biblioteki* jest ścieżką do IBM MQ classes for Java, która obejmuje biblioteki JNI. Tabela 52 na stronie 354 przedstawia położenie bibliotek produktu IBM MQ classes for Java dla każdej platformy. W tej tabeli *MQ_INSTALLATION_PATH* reprezentuje katalog najwyższego poziomu, w którym zainstalowano produkt IBM MQ .

Tabela 52. Położenie bibliotek produktu IBM MQ classes for Java dla każdej platformy	
Platforma	**Katalog zawierający biblioteki produktu IBM MQ classes for Java**
AIX	*MQ_INSTALLATION_PATH*/java/lib (biblioteki 32-bitowe) *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) *MQ_INSTALLATION_PATH*/java/lib64 (biblioteki 64-bitowe)
Solaris Solaris (platformyx86-64 i SPARC)	*MQ_INSTALLATION_PATH*/java/lib (biblioteki 32-bitowe) *MQ_INSTALLATION_PATH*/java/lib64 (biblioteki 64-bitowe)
Windows Windows	*MQ_INSTALLATION_PATH*\Java\lib (biblioteki 32-bitowe) *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. Solaris Linux AIX W systemach AIX, Linux (platforma Power), lub Solaris , należy użyć bibliotek 32-bitowych lub 64-bitowych. Biblioteki 64-bitowej należy używać tylko wtedy, gdy aplikacja jest uruchamiana w 64-bitowej maszynie wirtualnej Java (JVM) na platformie 64-bitowej. W przeciwnym razie należy użyć bibliotek 32-bitowych.
2. Windows W systemie Windowsmożna użyć zmiennej środowiskowej PATH w celu określenia położenia bibliotek produktu IBM MQ classes for Java zamiast określania ich położenia w komendzie **java** .
3. IBM i Aby używać produktu 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 korzystać z 32-bitowej lub 64-bitowej maszyny wirtualnej Java (JVM). Nie trzeba określać bibliotek, które mają być używane. IBM MQ classes for Java może 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 uruchamiała własne aplikacje.

Obsługa środowiska OSGi z produktem IBM MQ classes for Java
Środowisko OSGi udostępnia środowisko, które obsługuje wdrażanie aplikacji w postaci pakunków. Trzy pakunki OSGi są dostarczane jako część produktu IBM MQ classes for Java.

Środowisko OSGi udostępnia ogólne, bezpieczne i zarządzane środowisko produktu Java , które obsługuje wdrażanie aplikacji, które wchodzą w postaci pakunków. Urządzenia zgodne ze środowiskiem OSGi mogą pobierać i instalować pakunki, a także usuwać je, gdy nie są już wymagane. Środowisko zarządza instalowaniem i aktualizowaniem pakunków w sposób dynamiczny i skalowalny.

Produkt IBM MQ classes for Java zawiera następujące pakunki OSGi.

com.ibm.mq.osgi.java_*version_number*.jar
Pliki JAR, które umożliwiają aplikacjom korzystanie z produktu IBM MQ classes for Java.

com.ibm.mq.osgi.allclient_*version_number*.jar
Ten plik JAR umożliwia aplikacjom korzystanie zarówno z produktów IBM MQ classes for JMS , jak i IBM MQ classes for Java, a także dołącza kod do obsługi komunikatów PCF.

com.ibm.mq.osgi.allclientprereqs_*version_number*.jar
Ten plik JAR zawiera wymagania wstępne dla produktu `com.ibm.mq.osgi.allclient_version_number.jar`.

gdzie *version_number* jest numerem wersji produktu IBM MQ , który został zainstalowany.

Pakunki są instalowane w podkatalogu `java/lib/OSGi` w instalacji produktu 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 braku możliwości uruchamiania zarówno produktu IBM MQ classes for JMS , jak i produktu IBM MQ classes for Java w obrębie tych samych środowiska OSGi. Wszystkie pozostałe ograniczenia nadal obowiązują.
W przypadku wersji wcześniejszych niż IBM MQ 8.0zastosowanie ma ograniczenie dotyczące używania produktu IBM MQ classes for JMS lub IBM MQ classes for Java .

Dziewięć innych pakunków jest również zainstalowanych w podkatalogu `java/lib/OSGi` instalacji produktu IBM MQ lub w folderze `java\lib\OSGi` w systemie Windows. Pakunki te są częścią produktu IBM MQ classes for JMSi nie mogą być ładowane do środowiska wykonawczego OSGi, które ma załadowany pakunek IBM MQ classes for Java . Jeśli pakunek OSGi produktu IBM MQ classes for Java jest ładowany do środowiska wykonawczego OSGi, które zawiera również załadowane pakunki produktu IBM MQ classes for JMS , wówczas błędy przedstawione w poniższym przykładzie występują podczas uruchamiania aplikacji korzystających z pakunku IBM MQ classes for Java lub pakunków IBM MQ classes for JMS :

```
java.lang.ClassCastException: com.ibm.mq.MQException incompatible with com.ibm.mq.MQException
```

Pakunek OSGi dla produktu IBM MQ classes for Java został zapisany w specyfikacji OSGi Release 4. Nie działa on w środowisku OSGi Release 3.

Należy poprawnie ustawić ścieżkę systemową lub ścieżkę do biblioteki, aby środowisko wykonawcze OSGi było w stanie znaleźć wszystkie wymagane pliki DLL lub współużytkowane biblioteki.

Jeśli używany jest pakunek OSGi dla IBM MQ classes for Java, klasy wyjścia kanału napisane w produkcie Java nie są obsługiwane z powodu nieodłącznego problemu w ładowaniu klas w środowisku programu ładującego wiele klas, takim jak środowisko OSGi. Pakunek użytkownika może być świadomy pakunku IBM MQ classes for Java , ale pakunek IBM MQ classes for Java nie ma informacji o 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 można znaleźć w serwisie WWW OSGi alliance .

z/OS *Instalowanie produktu IBM MQ classes for Java w systemie z/OS*
W systemie z/OSbiblioteka STEPLIB używana w środowisku wykonawczym musi zawierać biblioteki IBM MQ SCSQAUTH i SCSQANLE.

W programie UNIX and Linux System Services można dodać te biblioteki, korzystając z wiersza w `.profile` , jak pokazano w poniższym przykładzie, zastępując `thlqual` kwalifikator zestawu danych wysokiego poziomu, który został wybrany podczas instalowania produktu IBM MQ:

```
export STEPLIB=thlqual.SCSQAUTH:thlqual.SCSQANLE:$STEPLIB
```

W innych środowiskach zwykle konieczne jest zmodyfikowanie uruchamiania JCL w taki sposób, aby uwzględniał on 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.

Format pliku konfiguracyjnego IBM MQ classes for Java jest taki sam jak standardowy plik właściwości Java .

Przykładowy plik konfiguracyjny `mqjava.config`jest dostarczany w podkatalogu `bin` katalogu instalacyjnego IBM MQ classes for Java . Ten plik dokumentuje wszystkie obsługiwane właściwości i ich wartości domyślne.

Uwaga: Przykładowy plik konfiguracyjny jest nadpisywany, gdy instalacja produktu IBM MQ zostanie zaktualizowana do przyszłego pakietu poprawek. Z tego powodu zaleca się, aby utworzyć kopię przykładowego pliku konfiguracyjnego do użycia z aplikacjami.

Użytkownik może wybrać nazwę i położenie pliku konfiguracyjnego produktu IBM MQ classes for Java . Po uruchomieniu aplikacji należy użyć komendy **java** z następującym formatem:

```
java -Dcom.ibm.msg.client.config.location=config_file_url application_name
```

W komendzie *config_file_url* jest jednostajnym lokalizatorem zasobów (URL), który określa 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 identyfikuje 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 przesłonić 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 z obsługiwanych transportów między aplikacją a menedżerem kolejek lub brokerem.

Nadpisywanie właściwości określonych w pliku konfiguracyjnym IBM MQ MQI client

Plik konfiguracyjny produktu 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 produktu IBM MQ MQI client mają zastosowanie tylko wtedy, gdy aplikacja łączy się z menedżerem kolejek w trybie klienta.

Jeśli jest to wymagane, można przesłonić dowolny atrybut w pliku konfiguracyjnym IBM MQ MQI client , określając go jako właściwość w pliku konfiguracyjnym IBM MQ classes for Java . Aby przesłonić atrybut w pliku konfiguracyjnym produktu IBM MQ MQI client , należy użyć wpisu o następującym formacie w pliku konfiguracyjnym produktu IBM MQ classes for Java :

```
com.ibm.mq.cfg.stanza.propName=propValue
```

Zmienne w pozycji mają następujące znaczenia:

sekcja
Nazwa sekcji w pliku konfiguracyjnym IBM MQ MQI client , który 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 przesłonić atrybut w pliku konfiguracyjnym IBM MQ MQI client , określając właściwość jako właściwość systemową w komendzie **java** . Aby określić właściwość jako właściwość systemową, należy użyć poprzedniego formatu.

Tylko następujące atrybuty w pliku konfiguracyjnym IBM MQ MQI client mają znaczenie dla produktu IBM MQ classes for Java. Jeśli zostaną określone lub nadpisane inne atrybuty, nie będzie to miało żadnego wpływu. W szczególności należy pamiętać, że `ChannelDefinitionFile` i `ChannelDefinitionDirectory` w sekcji CHANNELS w pliku konfiguracyjnym klienta nie są używane. Szczegółowe informacje na temat korzystania z tabeli definicji kanału klienta z IBM MQ classes for Javamożna znaleźć w sekcji “Korzystanie z tabeli definicji kanału klienta przy użyciu produktu IBM MQ classes for Java” na stronie 372 .

Tabela 53. Która sekcja pliku konfiguracyjnego klienta zawiera atrybut, który atrybut

sekcja	**Atrybut**
Sekcja CHANNELS pliku konfiguracyjnego klienta	Put1DefaultAlwaysSync
Sekcja CHANNELS pliku konfiguracyjnego klienta	PasswordProtection
ClientExitSekcja ścieżki pliku konfiguracyjnego klienta	ExitsDefaultPath
ClientExitSekcja ścieżki pliku konfiguracyjnego klienta	ExitsDefaultPath64
ClientExitSekcja ścieżki pliku konfiguracyjnego klienta	Ścieżka klasy JavaExits
Sekcja JMQI pliku konfiguracyjnego klienta	useMQCSPauthentication
Sekcja MessageBuffer pliku konfiguracyjnego klienta	MaximumSize
Sekcja MessageBuffer pliku konfiguracyjnego klienta	PurgeTime
Sekcja MessageBuffer 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 konfigurowania produktu IBM MQ MQI client zawiera sekcja Konfigurowanie klienta przy użyciu pliku konfiguracyjnego.

Zadania pokrewne

Śledzenie klas produktu IBM MQ dla aplikacji Java

Sekcja Java Standard Environment Trace

Aby skonfigurować narzędzie śledzenia produktu IBM MQ classes for Java , należy użyć sekcji Standard Trace Settings (Ustawienia śledzenia środowiska) produktu Java .

com.ibm.msg.client.commonservices.trace.outputName = *traceOutputNazwa*

traceOutputName to katalog i nazwa pliku, do którego wysyłane są 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 przypadku produktu IBM MQ classes for Java dla systemu IBM MQ 9.0.0 Fix Pack 1 lub wcześniejszej informacje śledzenia są zapisywane w pliku o nazwie `mqjms_%PID%.trc`.
- W produkcie IBM MQ 9.0.0 Fix Pack 2, jeśli aplikacja załadowała IBM MQ classes for Java z pliku JAR `com.ibm.mq.jar`, dane śledzenia są zapisywane w pliku o nazwie `mqjava_%PID%.trc`.
- W produkcie IBM MQ 9.0.0 Fix Pack 2, jeśli aplikacja załadowała 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`.
- V 9.1.5 V 9.1.0.5 W przypadku IBM MQ 9.1.5 i IBM MQ 9.1.0 Fix Pack 5, jeśli aplikacja załadowała 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`.
- V 9.1.5 V 9.1.0.5 W przypadku IBM MQ 9.1.5 i IBM MQ 9.1.0 Fix Pack 5, jeśli aplikacja załadowała 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`.

gdzie *%PID%* jest identyfikatorem procesu, który jest śledzony, a *%u* jest unikalną liczbą w celu odróżnienia plików między wątkami uruchamiających śledzenie w różnych programach ładujących klasy Java .

Jeśli identyfikator procesu jest niedostępny, generowany jest numer losowy, który jest poprzedzony literą `f`. Aby dołączyć identyfikator procesu do określonej nazwy pliku, należy użyć łańcucha %PID%.

Jeśli zostanie podany katalog alternatywny, musi on istnieć, a użytkownik musi mieć uprawnienia do zapisu dla tego katalogu. Jeśli użytkownik nie ma uprawnień do zapisu, dane wyjściowe śledzenia są zapisywane w produkcie `System.err`.

com.ibm.msg.client.commonservices.trace.include = *includeList*

includeList to lista śledzonych pakietów i klas albo wartości specjalne ALL lub NONE.

Oddziel nazwy pakietów lub klas średnikami, `;`. *includeList* przyjmuje wartości domyślne ALLi śledzi wszystkie pakiety i klasy w produkcie IBM MQ classes for Java.

Uwaga: Możliwe jest dołączenie pakietu, a następnie wykluczenie podpakietów tego pakietu. For example, if you include package `a.b` and exclude package `a.b.x`, the trace includes everything in `a.b.y` and `a.b.z`, but not `a.b.x` or `a.b.x.1`.

com.ibm.msg.client.commonservices.trace.exclude = *excludeList*

excludeList to lista pakietów i klas, które nie są śledzone, lub wartości specjalne ALL lub NONE.

Oddziel nazwy pakietów lub klas średnikami, `;`. Wartością domyślną *excludeList* jest NONE, dlatego nie jest wykluczany żaden pakiet i klasy w produkcie IBM MQ classes for JMS .

Uwaga: Użytkownik może wykluczyć pakiet, ale następnie uwzględnić podpakiety tego pakietu. Jeśli na przykład pakiet `a.b` zostanie wykluczony i dołączany jest pakiet `a.b.x`, dane śledzenia będą zawierały wszystkie elementy w produkcie `a.b.x` i `a.b.x.1`, ale nie będą zawierać wartości `a.b.y` ani `a.b.z`.

Uwzględniane są wszystkie pakiety lub klasy, które są określone na tym samym poziomie, co zarówno włączone, jak i wykluczone.

com.ibm.msg.client.commonservices.trace.maxBytes = *maxArrayBajty*
maxArrayBytes to maksymalna liczba bajtów, które są śledzone z dowolnej tablicy bajtów.

Jeśli parametr *maxArrayBytes* jest ustawiony na dodatnią liczbę całkowitą, ogranicza liczbę bajtów w tablicy bajtów, które są zapisywane w pliku śledzenia. Powoduje obcięcie tablicy bajtów po wypisaniu *maxArrayBytes* . Ustawienie *maxArrayBytes* redukuje wielkość wynikowego pliku śledzenia i zmniejsza wpływ śledzenia na wydajność aplikacji.

Wartość `0` dla tej właściwości oznacza, że żadna z treści żadnej tablicy bajtów nie jest wysyłana do pliku śledzenia.

Wartością domyślną jest `-1`, co powoduje usunięcie dowolnego 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 zapisanych w pliku wyjściowym śledzenia.

Produkt *maxTraceBytes* współpracuje z produktem *traceCycles*. Jeśli liczba zapisanych bajtów śledzenia znajduje się w pobliżu limitu, plik jest zamykany, a nowy plik wyjściowy śledzenia jest uruchamiany.

Wartość `0` oznacza, że plik wyjściowy śledzenia ma zerową długość. Wartością domyślną jest `-1`, co oznacza, że ilość danych, które mają zostać zapisane w pliku wyjściowym śledzenia, jest nieograniczona.

com.ibm.msg.client.commonservices.trace.count = *traceCycles*
traceCycles to liczba plików wyjściowych śledzenia, przez które mają zostać przełączone dane wyjściowe.

Jeśli bieżący plik wyjściowy śledzenia osiągnie limit określony przez parametr *maxTraceBytes*, plik zostanie zamknięty. Dalsze dane wyjściowe śledzenia są zapisywane w kolejnym pliku wyjściowym śledzenia w kolejności. Każdy plik wyjściowy śledzenia jest wyróżniany przyrostkiem liczbowym dodanym do nazwy pliku. Bieżący lub najnowszy plik wyjściowy śledzenia to `mqjms.trc.0`, następny najnowszy 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 parametr *traceCycles* ma wartość 1, wówczas gdy bieżący plik wyjściowy śledzenia osiągnie swoją maksymalną wielkość, plik jest zamykany i usuwany. Zostanie uruchomiony nowy plik wyjściowy śledzenia o tej samej nazwie. Oznacza to, że w danym momencie istnieje tylko jeden plik wyjściowy śledzenia.

com.ibm.msg.client.commonservices.trace.parameter = *traceParameters*
traceParameters określa, czy parametry metody i wartości zwracane są uwzględniane w śledzeniu.

Wartością domyślną *traceParameters* jest TRUE. Jeśli parametr *traceParameters* jest ustawiony na wartość FALSE, śledzone są tylko sygnatury metod.

com.ibm.msg.client.commonservices.trace.startup = *startup*
Jest to faza inicjowania IBM MQ classes for Java , podczas której przydzielane są zasoby. Główny narzędzie śledzenia jest inicjowany podczas fazy przydzielania zasobów.

Jeśli parametr *startup* jest ustawiony na wartość TRUE, używane jest śledzenie uruchamiania. Informacje śledzenia są generowane natychmiast i obejmują konfigurację wszystkich komponentów, w tym również funkcję śledzenia. Informacje śledzenia uruchamiania mogą być używane do diagnozowania problemów z konfiguracją. Informacje śledzenia uruchamiania są zawsze zapisywane w produkcie `System.err`.

Wartością domyślną *startup* jest FALSE.

startup jest sprawdzane przed ukończonainicjalizacją. Z tego powodu jako właściwość systemową Java należy określić właściwość w wierszu komend. Nie podauj go w pliku konfiguracyjnym IBM MQ classes for Java .

com.ibm.msg.client.commonservices.trace.compress = *compressedTrace*
Ustaw opcję *compressedTrace* na TRUE , aby skompresować dane wyjściowe śledzenia.

Wartością domyślną parametru *compressedTrace* jest FALSE.

Jeśli parametr *compressedTrace* jest ustawiony na wartość TRUE, dane wyjściowe śledzenia są kompresowane. Domyślna nazwa pliku wyjściowego śledzenia ma rozszerzenie `.trz`. Jeśli kompresja jest ustawiona na wartość FALSE(wartość domyślna), plik ma rozszerzenie `.trc` , aby wskazać, że jest on nieskompresowany. Jeśli jednak nazwa pliku dla danych wyjściowych śledzenia została określona w polu *traceOutputName* , to zamiast tego zostanie użyta nazwa pliku. Do tego pliku nie zostanie zastosowany przyrostek.

Skompresowane dane wyjściowe śledzenia są mniejsze niż nieskompresowane. Ponieważ istnieje mniej operacji we/wy, można je zapisać szybciej niż nieskompresowane śledzenie. Śledzenie skompresowane ma mniejszy wpływ na wydajność programu IBM MQ classes for Java niż nieskompresowane śledzenie.

Jeśli ustawione są opcje *maxTraceBytes* i *traceCycles* , to w miejsce wielu plików tekstowych tworzone są wiele skompresowanych plików śledzenia.

Jeśli produkt IBM MQ classes for Java zostanie zakończony w sposób niekontrolowany, skompresowany plik śledzenia może nie być poprawny. Z tego powodu kompresja śledzenia musi być używana tylko wtedy, gdy produkt IBM MQ classes for Java zostanie zamknięty w kontrolowany sposób. Kompresja śledzenia jest używana tylko wtedy, gdy badane problemy nie powodują nieoczekiwanego zatrzymania maszyny JVM. Nie należy używać kompresji śledzenia podczas diagnozowania problemów, które mogą spowodować zamknięcie systemu `System.Halt()` lub nieprawidłowe, niekontrolowane zakończenie 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`, to dowolny punkt śledzenia ze zdefiniowanym poziomem `TRACE_EXCEPTION`, `TRACE_WARNING`lub `TRACE_INFO` jest zapisywany w śledzeniu. Wszystkie pozostałe punkty śledzenia są wykluczone.

com.ibm.msg.client.commonservices.trace.standalone = *standaloneTrace*

standaloneTrace określa, czy usługa śledzenia klienta produktu IBM MQ classes for Java jest używana w środowisku produktu WebSphere Application Server .

Jeśli parametr *standaloneTrace* jest ustawiony na wartość TRUE, właściwości śledzenia klienta produktu IBM MQ classes for Java są używane do określenia konfiguracji śledzenia.

Jeśli parametr *standaloneTrace* jest ustawiony na 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.

Narzędzia IBM MQ classes for Java i narzędzia do zarządzania oprogramowaniem
Narzędzia do zarządzania oprogramowaniem, takie jak Apache Maven, mogą być używane razem z produktem IBM MQ classes for Java.

Wiele dużych organizacji programistycznych używa tych narzędzi do centralnego zarządzania repozytoriami bibliotek innych firm.

IBM MQ classes for Java składa się z wielu plików JAR. Podczas tworzenia aplikacji językowych produktu Java za pomocą tego interfejsu API, na komputerze, na którym aplikacja jest rozwijana, wymagana jest instalacja serwera IBM MQ , klienta IBM MQ lub klienta IBM MQ SupportPac .

Aby użyć narzędzia do zarządzania oprogramowaniem i dodać pliki JAR, które tworzą IBM MQ classes for Java w repozytorium zarządzanym centralnie, należy zaobserwować następujące punkty:

- Repozytorium lub kontener muszą być dostępne tylko dla programistów w organizacji. Dowolna dystrybucja poza organizacją jest niedozwolona.
- Repozytorium musi zawierać kompletny i spójny zestaw plików JAR z pojedynczego wydania produktu IBM MQ lub pakietu poprawek.
- Użytkownik jest odpowiedzialny za aktualizowanie repozytorium za pomocą obsługi technicznej udostępnianej przez dział wsparcia produktu IBM .

W produkcie IBM MQ 8.0do repozytorium musi być zainstalowany plik JAR produktu `com.ibm.mq.allclient.jar` .

Z poziomu produktu IBM MQ 9.0wymagane są pliki JAR obsługi zabezpieczeń Castle (Bouncy Castle) i pliki JAR obsługi CMS. Aby uzyskać więcej informacji, patrz “IBM MQ classes for Java relokacyjne pliki JAR” na stronie 349 i Wsparcie dla środowisk innych niżIBM JRE.

Konfigurowanie po instalacji dla aplikacji produktu IBM MQ classes for Java

Po zainstalowaniu produktu IBM MQ classes for Javamożna skonfigurować instalację w taki sposób, aby uruchamiała własne aplikacje.

Pamiętaj, aby sprawdzić plik readme produktu IBM MQ , aby uzyskać najnowsze informacje, lub aby uzyskać bardziej szczegółowe informacje na temat środowiska. Najnowsza wersja pliku readme produktu jest dostępna na stronie WWW produktu IBM MQ, WebSphere MQi MQSeries -pliki readme .

Przed podjęciem próby uruchomienia aplikacji IBM MQ classes for Java w trybie powiązań należy upewnić się, że skonfigurowano produkt IBM MQ zgodnie z opisem w sekcji Konfigurowanie.

Konfigurowanie menedżera kolejek w celu akceptowania połączeń klientów z produktu IBM MQ classes for Java
Aby skonfigurować menedżer kolejek w taki sposób, aby akceptować przychodzące żądania połączeń od klientów, należy zdefiniować i zezwolić na użycie kanału połączenia z serwerem i uruchomić program nasłuchujący.

Szczegółowe informacje można znaleźć w sekcji “Konfigurowanie menedżera kolejek w celu akceptowania połączeń klientów na wielu platformach” na stronie 1177.

Uruchamianie aplikacji produktu IBM MQ classes for Java w ramach Java security manager
Program IBM MQ classes for Java może działać z włączoną obsługą Java security manager . Aby pomyślnie uruchamiać aplikacje z włączoną obsługą Java security manager , należy skonfigurować Java virtual machine (JVM) z odpowiednim plikiem definicji strategii.

Najprostszym sposobem utworzenia odpowiedniego pliku definicji strategii jest zmiana pliku strategii dostarczanego razem z Java runtime environment (JRE). W większości systemów plik ten jest przechowywany w katalogu `path lib/security/java.policy`, który jest określony względem katalogu JRE. Pliki strategii można edytować przy użyciu preferowanego edytora lub za pomocą programu narzędziowego strategii dostarczanego ze środowiskiem JRE.

Użytkownik musi nadać uprawnienia do pliku `com.ibm.mq.jmqi.jar` , tak aby mógł:

- Tworzenie gniazd (w trybie klienta)
- Załaduj bibliotekę rodzimą (w trybie powiązań)

- Odczytywanie różnych właściwości z poziomu środowiska

Właściwość systemowa **os.name** musi być dostępna dla IBM MQ classes for Java podczas działania w Java security manager.

Jeśli aplikacja Java korzysta z Java security manager, należy dodać następujące uprawnienie do pliku `java.security.policy` używanego przez aplikację. W przeciwnym razie wyjątki zostaną zgłoszone do aplikacji:

```
permission java.lang.RuntimePermission "modifyThread";
```

Ten parametr RuntimePermission jest wymagany przez klienta jako część zarządzania przypisaniem i zamknięciem multipleksowanych konwersacji przez połączenia TCP/IP z menedżerami kolejek.

Przykładowa pozycja pliku strategii

Poniżej znajduje się przykład wpisu pliku strategii, który umożliwia IBM MQ classes for Java uruchamianie z powodzeniem w ramach domyślnego menedżera zabezpieczeń. Zastąp łańcuch *MQ_INSTALLATION_PATH* w tym przykładzie położeniem, w którym zainstalowano produkt IBM MQ classes for Java w systemie.

```
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 IBM MQ classes for Java poprawne działanie w ramach menedżera zabezpieczeń, ale może być konieczne włączenie własnego kodu w celu poprawnego działania przed pracą aplikacji.

Przykładowy kod dostarczany razem z produktem IBM MQ classes for Java nie został specjalnie włączony do użycia z menedżerem zabezpieczeń. Jednak testy programu IVT są uruchamiane z tym plikiem strategii i domyślnym menedżerem zabezpieczeń.

Uruchamianie aplikacji produktu IBM MQ classes for Java w produkcie CICS Transaction Server
Aplikacja IBM MQ classes for Java może być uruchamiana jako transakcja na serwerze CICS Transaction Server.

Aby uruchomić aplikację IBM MQ classes for Java jako transakcję na serwerze CICS Transaction Server for z/OS, wykonaj następujące kroki:

1. Zdefiniuj aplikację i transakcję na serwerze CICS , korzystając z podanej transakcji CEDA.
2. Upewnij się, że adapter IBM MQ CICS jest zainstalowany w systemie CICS . z/OS Szczegółowe informacje można znaleźć w sekcji Używanie produktu IBM MQ z produktem CICS .
3. Upewnij się, że środowisko JVM określone w produkcie CICS zawiera odpowiednie wpisy CLASSPATH i LIBPATH.
4. Zainicjuj transakcję, używając dowolnego z normalnych procesów.

Więcej informacji na temat uruchamiania transakcji CICS Java można znaleźć w dokumentacji systemu CICS .

Weryfikowanie instalacji produktu IBM MQ classes for Java

Program do weryfikacji instalacji, MQIVP, jest dostarczany razem z produktem IBM MQ classes for Java. Tego programu można użyć do przetestowania wszystkich trybów połączenia produktu IBM MQ classes for Java.

Program poprosi o podanie liczby wyborów i innych danych, aby określić, który tryb połączenia ma zostać sprawdzony. Aby sprawdzić instalację, wykonaj następującą procedurę:

1. Jeśli program ma być uruchamiany w trybie klienta, należy skonfigurować menedżer kolejek zgodnie z opisem w sekcji “Konfigurowanie menedżera kolejek w celu akceptowania połączeń klientów na wielu platformach” na stronie 1177. Kolejka, która ma być używana, to SYSTEM.DEFAULT.LOCAL.QUEUE.
2. Jeśli program ma być uruchamiany w trybie klienta, patrz także “użycieIBM MQ classes for Java” na stronie 343.

Wykonaj pozostałe kroki tej procedury w systemie, w którym ma zostać uruchomiony program.

3. Upewnij się, że zaktualizowano zmienną środowiskową CLASSPATH zgodnie z instrukcjami w sekcji “Zmienne środowiskowe istotne dla IBM MQ classes for Java” na stronie 352.
4. Zmień katalog na *MQ_INSTALLATION_PATH*/mqm/samp/wmqjava/samples, gdzie *MQ_INSTALLATION_PATH* jest ścieżką do instalacji produktu IBM MQ . Następnie w wierszu komend wpisz:

   ```
   java -Djava.library.path= library_path MQIVP
   ```

   gdzie *ścieżka_biblioteki* jest ścieżką do bibliotek produktu IBM MQ classes for Java (patrz “IBM MQ classes for Java biblioteki” na stronie 353).

   W pytaniu zaznaczonym (1):

   - Aby użyć połączenia TCP/IP, wprowadź nazwę hosta serwera IBM MQ .
   - Aby użyć połączenia rodzimego (tryb powiązań), należy pozostawić to pole puste (nie wpisuj nazwy).

   Program próbuje:

   - 1. Nawiąże połączenie z menedżerem kolejek
   - 2. Otwórz kolejkę SYSTEM.DEFAULT.LOCAL.QUEUE, umieść komunikat w kolejce, uzyskaj komunikat z kolejki, a następnie zamknij kolejkę.
   - 3. Odłączenie od menedżera kolejek
   - 4. Zwróć komunikat, jeśli operacje są pomyślne

Poniżej znajduje się przykład pytań i odpowiedzi, które można zobaczyć. Rzeczywiste zapytania i odpowiedzi zależą od sieci produktu 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 pole puste w wierszu komend oznaczonych (1).
2. Jeśli zostanie wybrane połączenie z serwerem, nie zostaną wyświetlone zachęty oznaczone jako (2).
3. IBM i W systemie IBM imożna tylko wydać komendę `java MQIVP` z QShell. Alternatywnie można uruchomić aplikację za pomocą komendy CL `RUNJVA CLASS(MQIVP)`.

Korzystanie z przykładowych aplikacji produktu IBM MQ classes for Java

Przykładowe aplikacje produktu IBM MQ classes for Java zawierają przegląd wspólnych funkcji interfejsu API produktu IBM MQ classes for Java . Można ich użyć do zweryfikowania konfiguracji serwera instalacji i przesyłania komunikatów oraz do pomocy przy tworzeniu własnych aplikacji.

O tym zadaniu

Jeśli potrzebna jest pomoc w tworzeniu własnych aplikacji, można użyć przykładowych aplikacji jako punktu początkowego. Zarówno źródło, jak i skompilowana wersja są udostępniane dla każdej aplikacji. Przejrzyj przykładowy kod źródłowy i zidentyfikuj kroki kluczowe, aby utworzyć każdy wymagany obiekt

dla aplikacji (MQQueueManager, MQConstants, MQMessage, MQPutMessagei MQDestination), a także aby ustawić konkretne właściwości, które są wymagane do określenia sposobu działania aplikacji. Więcej informacji na ten temat zawiera “Pisanie aplikacji produktu IBM MQ classes for Java” na stronie 368. Przykłady mogą ulec zmianie w przyszłych wersjach produktu IBM MQ Java.

Tabela 54 na stronie 365 pokazuje, gdzie przykładowe aplikacje produktu IBM MQ classes for Java są zainstalowane na każdej platformie:

Tabela 54. Katalogi instalacyjne dla przykładowych aplikacji produktu IBM MQ classes for Java

Platforma	**Katalog**
UNIX UNIX 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 55 na stronie 365 przedstawia zestawy przykładowych aplikacji, które są dostarczane razem z produktem IBM MQ classes for Java.

Tabela 55. IBM MQ classes for Java aplikacje przykładowe

Nazwa próbki	**Opis**
IMSBridgeSample.java	Prosty program demonstrujący przy użyciu mostu IMS z IBM MQ classes for Java.
MQIVP.java	Program weryfikujący instalację programu IBM MQ Java .
MQMessagePropertiesSample.java	Demonstruje użycie funkcji API Właściwości komunikatu wprowadzonej do produktu IBM WebSphere MQ 7.0.
MQPubSubApiSample.java	Demonstruje za pomocą funkcji API publikowania/subskrypcji wprowadzonej do produktu IBM WebSphere MQ 7.0.
MQSample.java	Prosty program w celu zademonstrować umieszczanie i pobieranie wiadomości z kolejki.
MQSampleMessageManager.java	Utility class for message handling in the IBM MQ base Java samples.
mqjcivp.properties	Ten pakunek zasobów zawiera komunikaty używane przez program weryfikujący instalację produktu IBM MQ classes for Java (`MQIVP.java`).

Produkt IBM MQ classes for Java udostępnia skrypt o nazwie `runjms` , który może być używany do uruchamiania przykładowych aplikacji. Ten skrypt konfiguruje środowisko produktu IBM MQ , aby umożliwić uruchamianie przykładowych aplikacji produktu IBM MQ classes for Java .

Tabela 56 na stronie 366 przedstawia położenie skryptu na każdej platformie:

Tabela 56. Położenie skryptu `runjms`	
Platforma	**Katalog**
UNIX UNIX 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 lub wersji /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. Przejdź do wiersza 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
   ```

   W przykładowej aplikacji wyświetlana jest lista 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, należy wprowadzić następujące komendy:

```
cd /opt/mqm/samp/wmqjava/samples
/opt/mqm/java/bin/runjms MQIVP
```

Pojęcia pokrewne

“Co jest zainstalowane w systemie IBM MQ classes for JMS” na stronie 87
Podczas instalowania produktu IBM MQ classes for JMStworzona jest pewna liczba plików i katalogów. W systemie Windowsniektóre konfiguracje są wykonywane podczas instalacji, automatycznie ustawiając zmienne środowiskowe. Na innych platformach i w niektórych środowiskach Windows należy ustawić zmienne środowiskowe, aby możliwe było uruchamianie aplikacji produktu IBM MQ classes for JMS .

Rozwiązywanie problemów z produktem IBM MQ classes for Java

Początkowo uruchom program weryfikujący instalację. Może być również konieczne użycie funkcji śledzenia.

Jeśli aplikacja nie zakończy się pomyślnie, uruchom program sprawdzający instalację i postępuj zgodnie z zaleceniami podanymi w komunikatach diagnostycznych. Program do weryfikacji instalacji jest opisany w sekcji “Weryfikowanie instalacji produktu IBM MQ classes for Java” na stronie 363.

Jeśli problemy są kontynuowane, a użytkownik musi skontaktować się z zespołem serwisowym IBM , może zostać poproszony o włączenie funkcji śledzenia. Należy to zrobić w sposób przedstawiony w poniższym przykładzie.

Aby śledzić program MQIVP :

- Utwórz plik właściwości produktu com.ibm.mq.commonservices (patrz sekcja Korzystanie z funkcji com.ibm.mq.commonservices.
- Wprowadź następującą komendę:

```
java -Dcom.ibm.mq.commonservices=commonservices_properties_file java
-Djava.library.path= library_path MQIVP -trace
```

gdzie:

- *plik_właściwości_commonservices_file* to ścieżka (wraz z nazwą pliku) do pliku właściwości produktu com.ibm.mq.commonservices .
- *ścieżka_biblioteki* to ścieżka do bibliotek produktu IBM MQ classes for Java (patrz sekcja “IBM MQ classes for Java biblioteki” na stronie 353).

Więcej informacji na temat sposobu korzystania ze śledzenia zawiera sekcja Śledzenie aplikacji IBM MQ classes for Java.

z/OS V 9.1.0 MQ Adv. VUE Połączenia klienta Java z aplikacjami wsadowymi działaniowymi w systemie z/OS

Za pomocą połączenia klienckiego aplikacja IBM MQ classes for Java na serwerze z/OS może łączyć się z menedżerem kolejek w systemie z/OS , który zawiera atrybut **ADVCAP**(ENABLED) . Korzystanie z połączenia klienckiego może uprościć topologie produktu IBM MQ .

Wartość **ADVCAP**(WŁĄCZONE) dotyczy tylko menedżera kolejek systemu z/OS , licencjonowanego jako IBM MQ Advanced for z/OS Value Unit Edition (patrz sekcja IBM MQ Identyfikatory produktów i informacje o eksporcie) i działającego z wartością **QMGRPROD** ustawioną na ADVANCEDVUE.

Patrz DISPLAY QMGR , aby uzyskać więcej informacji na temat **ADVCAP** i START QMGR , aby uzyskać więcej informacji na temat **QMGRPROD**.

Aplikacja IBM MQ classes for Java w systemie z/OS nie może korzystać z połączenia w trybie klienta z menedżerem kolejek, który nie jest uruchomiony na serwerze z/OS, lub z menedżerem kolejek, który nie ma ustawionego zestawu opcji **ADVCAP**(ENABLED) .

Jeśli aplikacja IBM MQ classes for Java w systemie z/OS próbuje nawiązać połączenie przy użyciu trybu klienta i nie jest to dozwolone, zwracana jest wartość MQRC_ENVIRONMENT_ERROR .

Obsługa Advanced Message Security (AMS)

V 9.1.0

From IBM MQ 9.1, IBM MQ classes for Java client applications can use AMS when connecting to IBM MQ Advanced for z/OS Value Unit Edition queue managers on remote z/OS systems.

Nowy typ magazynu kluczy, jceracfks, jest obsługiwany tylko w systemie keystore.conf na platformie z/OS , 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
```

Pisanie aplikacji produktu IBM MQ classes for Java

Ta kolekcja tematów zawiera informacje pomocne w pisaniu aplikacji produktu Java w celu interakcji z systemami IBM MQ .

To use IBM MQ classes for Java to access IBM MQ queues, you write Java applications that contain calls that put messages onto, and get messages from, IBM MQ queues. Szczegółowe informacje na temat poszczególnych klas można znaleźć w sekcji 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

W proceduralnym interfejsie programistycznym aplikacji IBM MQ używane są czasowniki, które działają na obiektach. Interfejs programistyczny Java używa obiektów, które są używane przy użyciu metod wywołującej.

Proceduralny interfejs programistyczny aplikacji IBM MQ jest zbudowany wokół czasowników, takich jak:

```
MQBACK, MQBEGIN, MQCLOSE, MQCONN, MQDISC,
MQGET, MQINQ, MQOPEN, MQPUT, MQSET, MQSUB
```

Te czasowniki przyjmują, jako parametr, uchwyt do obiektu IBM MQ , na którym mają działać. Program składa się z zestawu obiektów IBM MQ , które są używane przez wywołania metod na tych obiektach.

Podczas korzystania z interfejsu proceduralnego następuje rozłączenie z menedżerem kolejek przy użyciu wywołania MQDISC (Hconn, CompCode, Reason), gdzie *Hconn* jest uchwytem dla menedżera kolejek.

W interfejsie Java menedżer kolejek jest reprezentowany przez obiekt klasy MQQueueManager. Odłącz się od menedżera kolejek, wywołując metodę 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, w jaki program IBM MQ classes for Java ma pewne zależności od trybów połączenia, które mają być używane.

Jeśli używane są połączenia klienckie, istnieje wiele różnic między IBM MQ MQI client , ale jest to koncepcyjnie podobne. Jeśli używany jest tryb powiązań, można użyć powiązań krótkiej ścieżki i można wywołać komendę MQBEGIN. Należy określić tryb, który ma być używany przez ustawienie zmiennych w klasie MQEnvironment.

IBM MQ classes for Java połączenia klientów

Gdy produkt IBM MQ classes for Java jest używany jako klient, jest on podobny do IBM MQ MQI client, ale ma wiele różnic.

Jeśli programowanie dla *IBM MQ classes for Java* jest przeznaczone do użytku jako klient, należy pamiętać o następujących różnicach:

- Obsługuje on tylko protokół TCP/IP.
- Podczas uruchamiania nie odczytuje żadnych zmiennych środowiskowych IBM MQ .
- Informacje, które będą przechowywane w definicji kanału oraz w zmiennych środowiskowych, mogą być przechowywane w klasie o nazwie Środowisko. Alternatywnie, informacje te mogą być przekazywane jako parametry podczas nawiązywania połączenia.
- Warunki błędu i wyjątku są zapisywane w dzienniku określonym w klasie `MQException` . Domyślnym miejscem docelowym błędów jest konsola produktu Java .
- Tylko następujące atrybuty w pliku konfiguracyjnym klienta IBM MQ mają znaczenie dla produktu IBM MQ classes for Java. Jeśli zostaną określone inne atrybuty, są one nieskuteczne.

sekcja	**Atrybut**
ClientExitSekcja ścieżki pliku konfiguracyjnego klienta	ExitsDefaultPath
ClientExitSekcja ścieżki pliku konfiguracyjnego klienta	ExitsDefaultPath64
ClientExitSekcja ścieżki pliku konfiguracyjnego klienta	JavaExitsClasspath
Sekcja MessageBuffer pliku konfiguracyjnego klienta	MaximumSize
Sekcja MessageBuffer pliku konfiguracyjnego klienta	PurgeTime
Sekcja MessageBuffer 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

- W przypadku łączenia się z menedżerem kolejek, który wymaga konwersji danych znakowych, klient V7 Java jest teraz w stanie wykonać konwersję, jeśli menedżer kolejek nie jest w stanie wykonać tej operacji. Maszyna JVM klienta musi obsługiwać konwersję między identyfikatorem CCSID klienta a tym menedżerem kolejek.
- Automatyczne ponowne nawiązywanie połączenia przez klient nie jest obsługiwane przez produkt IBM MQ classes for Java.

W przypadku użycia w trybie klienta program *IBM MQ classes for Java* nie obsługuje wywołania MQBEGIN.

Tryb powiązań IBM MQ classes for Java
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 interfejsu JNI (Java Native Interface) do wywołania bezpośrednio do istniejącego interfejsu API menedżera kolejek, zamiast komunikowania się przez sieć.

Domyślnie aplikacje korzystające z produktu IBM MQ classes for Java w trybie powiązań łączą się z menedżerem kolejek przy użyciu opcji *ConnectOption*(MQCNO_STANDARD_BINDINGS).

Produkt IBM MQ classes for Java obsługuje następujące elementy *ConnectOptions*:

- MQCNO_FASTPATH_BINDING
- MQCNO_STANDARD_BINDING
- MQCNO_SHARED_BINDING

- MQCNO_ISOLATED_BINDING

Więcej informacji na temat opcji *ConnectOptions*można znaleźć w sekcji “Nawiąże połączenie z menedżerem kolejek przy użyciu wywołania MQCONNX” na stronie 827.

Tryb powiązań obsługuje wywołanie komendy MQBEGIN w celu zainicjowania globalnych jednostek pracy, które są koordynowane przez menedżer kolejek, na wszystkich platformach poza produktem IBM MQ for IBM i i IBM MQ for z/OS.

Większość parametrów udostępnianych przez klasę `MQEnvironment` nie jest odpowiednia dla trybu powiązań i są ignorowane.

Definiowanie, które połączenie produktu IBM MQ classes for Java ma zostać użyte
Typ połączenia, który ma być używany, jest określany przez ustawienie zmiennych w klasie MQEnvironment.

Używane są dwie zmienne:

MQEnvironment.properties
Typ połączenia jest określany na podstawie wartości powiązanej 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 na podstawie wartości 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 przypadku trybu powiązań nie należy ustawiać tej zmiennej ani ustawić jej na wartość NULL.

Operacje na menedżerach kolejek

W tej kolekcji tematów opisano sposób łączenia i rozłączenia menedżera kolejek za pomocą programu IBM MQ classes for Java.

Konfigurowanie środowiska IBM MQ dla produktu IBM MQ classes for Java
W przypadku aplikacji w celu nawiązania połączenia z menedżerem kolejek w trybie klienta aplikacja musi określić nazwę kanału, nazwę hosta i numer portu.

Uwaga: Informacje zawarte w tym temacie są istotne tylko wtedy, gdy aplikacja łączy się z menedżerem kolejek w trybie klienta. Nie ma znaczenia, czy łączy się ona w trybie powiązań. Patrz sekcja “Tryby połączenia dla IBM MQ classes for JMS” na stronie 107.

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 pola zostaną ustawione w klasie MQEnvironment, będą one stosowane do całej aplikacji, z wyjątkiem sytuacji, w których są one 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 równoznaczne 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 zostaną przekazane do obiektu menedżera kolejek przy jego utworzeniu, będą one miały zastosowanie tylko do tego menedżera kolejek. Tworzenie wpisów 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, która identyfikuje jego połączenie z menedżerem kolejek. Ta nazwa aplikacji jest wyświetlana za pomocą komendy **DISPLAY CONN MQSC/PCF** (gdzie pole to jest nazywane **APPLTAG**) lub na ekranie **Połączenia aplikacji** programu IBM MQ Explorer (gdzie pole to jest nazywane **App name**).

Nazwy aplikacji są ograniczone do 28 znaków, tak więc dłuższe nazwy są obcinane. Jeśli nazwa aplikacji nie jest określona, zostanie podana wartość domyślna. Nazwa domyślna jest oparta na klasie wywołującej (main), ale jeśli te informacje nie są dostępne, używany jest tekst `IBM MQ Client for Java` .

Jeśli używana jest nazwa klasy wywołującej, jest ona dopasowywana w taki sposób, aby pasować, usuwając początkowe nazwy pakietów, jeśli to konieczne. Na przykład, jeśli klasą wywołującym jest `com.example.MainApp`, używana jest pełna nazwa, ale jeśli klasą wywołującym jest `com.example.dictionaryAndThesaurus.multilingual.mainApp`, używana jest nazwa `multilingual.mainApp` , ponieważ jest to najdłuższa kombinacja nazwy klasy i najbardziej należnej nazwy pakietu, która mieści się w dostępnej długości.

Jeśli sama nazwa klasy ma więcej niż 28 znaków, zostanie obcięta do dopasowania. Na przykład `com.example.mainApplicationForSecondTestCase` staje się `mainApplicationForSecondTest`.

Aby ustawić nazwę aplikacji w klasie MQEnvironment, należy dodać nazwę do tabeli mieszającej MQEnvironment.properties z kluczem **MQConstants.APPNAME_PROPERTY**przy użyciu 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 produktu IBM MQ MQI client mają zastosowanie tylko wtedy, gdy aplikacja łączy się z menedżerem kolejek w trybie klienta.

Jeśli jest to wymagane, można przesłonić dowolny atrybut w pliku konfiguracyjnym IBM MQ w jeden z następujących sposobów. Opcje są wyświetlane w kolejności wykonywania.

- Ustaw właściwość systemową Java dla właściwości konfiguracyjnej.
- Ustaw właściwość w odwzorowaniu MQEnvironment.properties .
- W systemie Java5 i nowszych wersjach ustaw systemową zmienną środowiskową.

Tylko następujące atrybuty w pliku konfiguracyjnym klienta IBM MQ mają znaczenie dla produktu IBM MQ classes for Java. Jeśli zostaną określone lub nadpisane inne atrybuty, nie będzie to miało żadnego wpływu.

sekcja	**Atrybut**
ClientExitSekcja ścieżki pliku konfiguracyjnego klienta	ExitsDefaultPath
ClientExitSekcja ścieżki pliku konfiguracyjnego klienta	ExitsDefaultPath64
ClientExitSekcja ścieżki pliku konfiguracyjnego klienta	JavaExitsClasspath
Sekcja MessageBuffer pliku konfiguracyjnego klienta	MaximumSize
Sekcja MessageBuffer pliku konfiguracyjnego klienta	PurgeTime
Sekcja MessageBuffer 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

Nawiąże połączenie z menedżerem kolejek w produkcie IBM MQ classes for Java
Połącz się z menedżerem kolejek, tworząc nową instancję klasy MQQueueManager . Odłącz się od menedżera kolejek, wywołując metodę rozłączania ().

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 rozłączania, wszystkie otwarte kolejki i procesy, do których użytkownik uzyskuje dostęp za pośrednictwem tego menedżera kolejek, są zamykane. Jednak dobrym rozwiązaniem programowym jest zamknięcie tych zasobów w sposób jawny po zakończeniu korzystania z nich. Aby to zrobić, należy użyć metody close () dla odpowiednich obiektów.

Metody commit () i backout () w menedżerze kolejek są równoważne wywołaniach MQCMIT i MQBACK, które są używane w interfejsie proceduralnym.

Korzystanie z tabeli definicji kanału klienta przy użyciu produktu IBM MQ classes for Java
Aplikacja kliencka IBM MQ classes for Java może korzystać z definicji kanału połączenia klienta zapisanych w tabeli definicji kanału klienta (CCDT).

Alternatywą dla utworzenia definicji kanału połączenia klienta przez ustawienie określonych pól i właściwości środowiska w klasie `MQEnvironment` lub przekazanie ich do `MQQueueManager` w tabeli

mieszającej właściwości aplikacja kliencka IBM MQ classes for Java może używać definicji kanału połączenia klienta zapisanych w tabeli definicji kanału klienta. Definicje te są tworzone za pomocą komend IBM MQ Script (MQSC) lub IBM MQ Programmable Command Format (PCF) lub za pomocą IBM MQ Explorer .

Gdy aplikacja tworzy obiekt `MQQueueManager` , klient IBM MQ classes for Java przeszukuje tabelę definicji kanału klienta pod kątem odpowiedniej definicji kanału połączenia klienckiego i używa definicji kanału w celu uruchomienia kanału MQI. Więcej informacji na temat tabel definicji kanału klienta i sposobu ich tworzenia zawiera sekcja Tabela definicji kanału klienta.

Aby użyć tabeli definicji kanału klienta, aplikacja musi najpierw utworzyć obiekt URL. Obiekt URL hermetyzuje jednolity wskaźnik zasobów (URL), który identyfikuje nazwę i położenie pliku zawierającego tabelę definicji kanału klienta i określa sposób dostępu do pliku.

Na przykład, jeśli 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");
```

Załóżmy na przykład, 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 obiektu adresu URL aplikacja może utworzyć obiekt `MQQueueManager` , korzystając z jednego z konstruktorów, który pobiera obiekt URL jako parametr. 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 adresu URL chanTab2, wyszuka odpowiednią definicję kanału połączenia klienckiego, a następnie używa definicji kanału w celu uruchomienia kanału MQI dla menedżera kolejek o nazwie MARS.

Należy zwrócić uwagę na następujące punkty, które mają zastosowanie, jeśli aplikacja korzysta z tabeli definicji kanału klienta:

- Gdy aplikacja tworzy obiekt `MQQueueManager` przy użyciu konstruktora, który pobiera obiekt URL jako parametr, żadna nazwa kanału nie musi być ustawiona w klasie `MQEnvironment` jako właściwość w polu lub jako właściwość środowiska. Jeśli nazwa kanału jest ustawiona, klient IBM MQ classes for Java zgłasza `MQException`. Właściwość pola lub środowiska określająca nazwę kanału jest uznawana za ustawioną, jeśli jej wartością jest dowolna wartość inna niż null, pusty łańcuch lub łańcuch zawierający wszystkie puste znaki.
- Parametr **queueManagerName** w konstruktorze `MQQueueManager` może mieć jedną z następujących wartości:
  - Nazwa menedżera kolejek
  - Gwiazdka (*), po której następuje nazwa grupy menedżerów kolejek
  - Gwiazdka (*)
  - Wartość NULL, pusty łańcuch lub łańcuch zawierający wszystkie puste znaki.

  Są to te same wartości, których można użyć dla parametru **QMgrName** w wywołaniu MQCONN wywoływanym przez aplikację kliencką korzystaniową z interfejsu kolejki komunikatów (Message Queue Interface-MQI). Więcej informacji na temat znaczenia tych wartości znajduje się w sekcji“Interfejs kolejki komunikatów-przegląd” na stronie 811.

  Jeśli aplikacja korzysta z zestawiania połączeń, należy zapoznać się z“Sterowanie domyślną pulą połączeń w programie IBM MQ classes for Java” na stronie 393.

- 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 w celu 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 w szczególności zwrócić uwagę na następujące kwestie:

  – Kanał MQI używa protokołu TLS tylko wtedy, gdy definicja kanału wyodrębniona z tabeli definicji kanału klienta określa nazwę klasy CipherSpec obsługiwaną przez klient IBM MQ classes for Java .
  – Tabela definicji kanału klienta zawiera również informacje na temat położenia serwerów LDAP (Lightweight Directory Access Protocol), które przechowują listy odwołań certyfikatów (CRL). Klient IBM MQ classes for Java używa tylko tych informacji w celu uzyskania dostępu do serwerów LDAP, które przechowują listy CRL.
  – Tabela definicji kanału klienta może również zawierać położenie respondera OCSP. Produkt IBM MQ classes for Java nie może używać informacji OCSP z pliku tabeli definicji kanału klienta. Można jednak skonfigurować protokół OCSP zgodnie z opisem w sekcji Korzystanie z protokołu certyfikatów online .

  Więcej informacji na temat używania protokołu TLS przy użyciu tabeli definicji kanału klienta zawiera sekcja Określanie, czy kanał MQI używa protokołu TLS.

  Jeśli używane są wyjścia kanału, należy pamiętać również o następujących punktach:

  – Kanał MQI używa wyjść kanału i powiązanych danych użytkownika określonych przez definicję kanału wyodrębnioną z tabeli definicji kanału klienta w preferencjach do wyjść kanału i danych określonych przy użyciu innych metod.
  – Definicja kanału wyodrębniona z tabeli definicji kanału klienta może określać wyjścia kanału, które są zapisywane w produkcie Java, C lub C + +. Więcej informacji na temat pisania wyjścia kanału w programie Java zawiera sekcja“Tworzenie wyjścia kanału w produkcie IBM MQ classes for Java” na stronie 387. Więcej informacji na temat pisania wyjścia kanału w innych językach zawiera sekcja “Używanie wyjść kanału, które nie zostały zapisane w programie Java z programem IBM MQ classes for Java” na stronie 391.

Określanie zakresu portów dla połączeń klienta IBM MQ classes for Java
Można określić port lub zakres portów, które mogą być powiązane z aplikacją na jeden z dwóch sposobów.

Gdy aplikacja IBM MQ classes for Java próbuje nawiązać połączenie z menedżerem kolejek produktu IBM MQ w trybie klienta, firewall może zezwalać tylko na połączenia, które pochodzą z określonych portów lub zakresu portów. W takiej sytuacji można określić port lub zakres portów, z którymi aplikacja może się wiązać. Port (y) można określić w następujący sposób:

- Pole localAddress(Adres lokalny) można ustawić w klasie MQEnvironment. Oto przykład:

```
MQEnvironment.localAddressSetting = "192.0.2.0(2000,3000)";
```

- Istnieje możliwość ustawienia właściwości środowiska CMQC.LOCAL_ADDRESS_PROPERTY. Oto przykład:

```
(MQEnvironment.properties).put(CMQC.LOCAL_ADDRESS_PROPERTY,
                               "192.0.2.0(2000,3000)");
```

- Jeśli można skonstruować obiekt MQQueueManager , można przekazać właściwości hashtable zawierające właściwość LOCAL_ADDRESS_PROPERTY o wartości "192.0.2.0(2000,3000)".

W każdym z tych przykładów, gdy aplikacja nawiąże później połączenie z menedżerem kolejek, aplikacja wiąże 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 localAddressSetting (Adres lokalny) lub właściwości środowiska CMQC.LOCAL_ADDRESS_PROPERTY, aby określić, który interfejs sieciowy musi być używany dla połączenia.

W przypadku ograniczenia zakresu portów mogą wystąpić błędy połączenia. Jeśli wystąpi błąd, zgłaszany jest wyjątek MQException zawierający kod przyczyny produktu IBM MQ MQRC_Q_MGR_NOT_AVAILABLE i następujący komunikat:

```
Socket connection attempt refused due to LOCAL_ADDRESS_PROPERTY restrictions
```

Błąd może wystąpić, jeśli używane są wszystkie porty w podanym zakresie lub jeśli podany adres IP, nazwa hosta lub numer portu nie są poprawne (na przykład jest to 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 . Tabela MQOD (struktura deskryptora obiektu) jest zwinięta do parametrów tych metod.

Kolejki

Aby otworzyć kolejkę, można użyć metody accessQueue klasy MQQueueManager . Na przykład w menedżerze 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 należy użyć metody close (), aby ją zamknąć, tak jak w następującym przykładzie:

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

Kolejkę można również utworzyć za pomocą konstruktora MQQueue. Parametry są dokładnie takie same, jak w przypadku metody accessQueue , z dodaniem parametru menedżera kolejek. Na przykład:

```
MQQueue queue = new MQQueue(queueManager,
                            "qName",
                            CMQC.MQOO_OUTPUT,
                            "qMgrName",
                            "dynamicQName",
                            "altUserID");
```

W przypadku tworzenia kolejek można określić wiele opcji. Szczegółowe informacje na ten temat można znaleźć w sekcji Class.com.ibm.mq.MQQueue. Konstruowanie obiektu kolejki w ten sposób umożliwia zapisywanie własnych podklas kolejki MQQueue.

Tematy

W podobny sposób 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 należy użyć metody close (), aby ją zamknąć.

Istnieje również możliwość utworzenia tematu 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. Konstruowanie obiektu tematu w ten sposób umożliwia pisanie własnych podklas tematu MQTopic.

Temat musi zostać otwarty albo w celu opublikowania, albo w celu 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). Mogą one być używane tylko do otwierania tematu dla subskrypcji. Dwie pozostałe metody mają parametr **openAs** , a temat może zostać otwarty 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, a w obu przypadkach należy ustawić wartość CMQC.MQSO_DURABLE .

Procesy

Aby uzyskać dostęp do procesu, należy użyć metody accessProcess obiektu MQQueueManager. Na przykład w menedżerze kolejek o nazwie queueManagernależy użyć następującego kodu w celu utworzenia obiektu MQProcess:

```
MQProcess process =
queueManager.accessProcess("PROCESSNAME",
CMQC.MQOO_FAIL_IF_QUIESCING);
```

Aby uzyskać dostęp do procesu, należy użyć metody accessProcess obiektu MQQueueManager.

Metoda accessProcess zwraca nowy obiekt MQProcess klasy.

Po zakończeniu korzystania z obiektu procesu należy użyć metody close (), aby ją zamknąć, tak jak w następującym przykładzie:

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

Proces można również utworzyć za pomocą 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);
```

Konstruowanie obiektu procesu w ten sposób umożliwia pisanie własnych podklas procesu MQProcess.

Obsługa komunikatów w produkcie IBM MQ classes for Java

Komunikaty są reprezentowane przez klasę MQMessage. Komunikaty są umieszczane i wysyłane przy użyciu metod klasy MQDestination, które mają podklasy MQQueue i MQTopic.

Umieszczanie komunikatów w kolejkach lub tematach przy użyciu metody put () klasy MQDestination. Komunikaty z kolejek lub tematów są wysyłane za pomocą metody get () klasy MQDestination. W przeciwieństwie do interfejsu proceduralnego, w którym są umieszczane tablice MQPUT i MQGET i otrzymujemy tablice bajtów, język programowania produktu Java umieszcza i pobiera instancje klasy MQMessage. Klasa MQMessage hermetykuje bufor danych, który zawiera rzeczywiste dane komunikatu,

wraz ze wszystkimi parametrami deskryptora MQMD (deskryptor komunikatu) oraz właściwościami komunikatu opisujących ten komunikat.

Aby zbudować nowy komunikat, należy utworzyć nową instancję klasy MQMessage i użyć metod writeXXX w celu umieszczenia danych w buforze komunikatów.

Po utworzeniu nowej instancji komunikatu wszystkie parametry MQMD są automatycznie ustawiane na wartości domyślne, zgodnie z definicją w sekcji Wartości początkowe i deklaracje języków dla deskryptora MQMD. Metoda put () obiektu MQDestination pobiera również instancję klasy opcji MQPutMessagejako parametr. Ta klasa reprezentuje strukturę MQPMO. W poniższym przykładzie tworzony jest 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 komunikatu MQMessage, która reprezentuje komunikat, który został właśnie odebrany z kolejki. Pobiera ona również instancję klasy opcji MQGetMessagejako parametr. Ta klasa reprezentuje strukturę MQGMO.

Nie ma potrzeby określania maksymalnej wielkości komunikatu, ponieważ metoda get () automatycznie dostosowuje wielkość swojego wewnętrznego buforu, aby pasował do komunikatu przychodzącego. Aby uzyskać dostęp do danych w zwróconej wiadomości, należy użyć metod readXXX klasy MQMessage.

W poniższym przykładzie przedstawiono sposób pobrania 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);
```

Istnieje możliwość zmiany formatu liczb, którego używają metody odczytu i zapisu, ustawiając zmienną elementu *encoding* .

Zestaw znaków używany do odczytu i zapisu łańcuchów można zmienić, ustawiając zmienną elementu *characterSet* .

Więcej informacji zawiera sekcja “Klasa MQMessage” na stronie 724.

Uwaga: Metoda writeUTF() obiektu MQMessage automatycznie koduje długość łańcucha, a także bajty Unicode, które zawiera. Gdy komunikat zostanie odczytany przez inny program Java (za pomocą komendy readUTF()), jest to najprostszy sposób wysyłania informacji łańcuchowych.

Zwiększanie wydajności komunikatów nietrwałych w produkcie IBM MQ classes for Java
Aby zwiększyć wydajność podczas przeglądania komunikatów lub korzystania z nietrwałych komunikatów z aplikacji klienckiej, można użyć funkcji *read ahead*(odczyt z wyprzedzeniem). Aplikacje klienckie używające wywołania MQGET lub asynchronicznego będą korzystać z usprawnień wydajności podczas przeglądania komunikatów lub korzystania z nietrwałych komunikatów.

Ogólne informacje na temat narzędzia do odczytu z wyprzedzeniem zawiera sekcja pokrewny temat.

W produkcie IBM MQ classes for Javaużywana jest opcja CMQC.MQSO_READ_AHEAD i CMQC.MQSO_NO_READ_AHEAD obiektu MQQueue lub obiektu MQTopic w celu określenia, czy konsumenci komunikatów i przeglądarki kolejek mogą używać odczytu z wyprzedzeniem dla tego obiektu.

Asynchronicznie umieszczanie komunikatów przy użyciu produktu IBM MQ classes for Java
Aby umieścić komunikat asynchronicznie, należy ustawić wartość MQPMO_ASYNC_RESPONSE.

Komunikaty umieszczane są w kolejkach lub tematach przy użyciu metody put () klasy MQDestination. Aby umieścić komunikat asynchronicznie, to znaczy zezwalając na zakończenie operacji bez oczekiwania na odpowiedź z menedżera kolejek, można ustawić wartość MQPMO_ASYNC_RESPONSE w polu opcji opcji MQPutMessage. Aby określić powodzenie lub niepowodzenie operacji put asynchronicznych, 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, a użytkownik opublikuje go przy użyciu metod MQTopic.put().

Ogólne informacje na temat publikowania/subskrypcji produktu IBM MQ można znaleźć w temacie Przesyłanie 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 za pomocą IBM MQ classes for Java
W tym przykładzie instancja obiektu MQHeaderIterator analizuje nagłówki w komunikacie MQMessage, który został odebrany z kolejki. Obiekty MQHeader zwracane przez metodę nextHeader() wyświetlają ich strukturę i zawartość, gdy wywoływana jest metoda 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 za pomocą IBM MQ classes for Java
W tym przykładzie metoda skipHeaders() klasy MQHeaderIterator umieszcza kursor odczytu komunikatu bezpośrednio po ostatnim nagłówku.

```
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 programu IBM MQ classes for Java
W tym przykładzie metoda read zapełnia obiekt MQDLH odczytem z komunikatu. Po operacji odczytu kursor odczytu komunikatu jest ustawiany natychmiast po treści nagłówka MQDLH.

Komunikaty w kolejce niedostarczonych komunikatów menedżera kolejek są poprzedzane nagłówkiem z niedostarczonym literem (MQDLH). Aby zdecydować, w jaki sposób obsługiwać te komunikaty-na przykład w celu określenia, czy mają być one ponawiane, czy je odrzucić-w aplikacji obsługi niedostarczonych komunikatów należy sprawdzić kod przyczyny znajdujący się w tabeli 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 wygodny konstruktor, który inicjuje się bezpośrednio z komunikatu w jednym 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 niedostarczonych komunikatów za pomocą programu IBM MQ classes for Java
W tym przykładzie komenda MQDLH jest używana do usuwania nagłówka z komunikatu niewysłanych wiadomości.

Aplikacja obsługi niedostarczonych komunikatów będzie zazwyczaj ponownie wysyłać komunikaty, które zostały odrzucone, jeśli ich kod przyczyny wskazuje na błąd przejściowy. Przed ponownym przesłaniem komunikatu musi on usunąć nagłówek MQDLH.

W tym przykładzie wykonywane są następujące kroki (patrz komentarze w kodzie przykładowym):

1. Element MQHeaderList odczytuje całą wiadomość, a każdy nagłówek napotkany w komunikacie staje się elementem na liście.
2. Komunikaty o niewysłanych literach zawierają kod MQDLH jako pierwszy nagłówek, więc można go znaleźć w pierwszym elemencie listy nagłówków. Obiekt MQDLH został już zapełniony z komunikatu, gdy jest zbudowana lista MQHeaderList , dlatego nie ma potrzeby wywoływania metody odczytu.
3. Kod przyczyny jest wyodrębniany przy użyciu metody getReason() udostępnianej przez klasę MQDLH.
4. Kod przyczyny został sprawdzony i wskazuje, że należy wprowadzić ponownie komunikat. Zmaterializowana tabela MQDLH jest usuwana przy użyciu metody remove () MQHeaderList .
5. Obiekt MQHeaderList zapisuje swoją pozostałą treść w nowym obiekcie komunikatu. Nowy komunikat zawiera teraz wszystko, co znajduje się w oryginalnym komunikacie, z wyjątkiem MQDLH i może zostać zapisany w kolejce. Argument **true** dla konstruktora i metody zapisu wskazuje, że treść komunikatu ma być wstrzymana w obrębie elementu MQHeaderListi jest zapisywana ponownie.
6. Pole formatu w deskryptorze komunikatu nowego komunikatu zawiera teraz wartość, która była wcześniej określona w polu formatu MQDLH. Dane komunikatu są zgodne z kodowaniem liczbowym i zestawem 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 w celu wydrukowania treści komunikatu, w tym jego nagłówków.

Dane wyjściowe zawierają widok wszystkich treści nagłówka, a także treści komunikatu. Klasa MQHeaderList dekoduje wszystkie nagłówki w jednym kroku, podczas gdy iterator MQHeaderIterator wykonuje je po jednym w czasie pod kontrolą aplikacji. Tej techniki można użyć do udostępnienia prostego narzędzia do debugowania podczas zapisywania aplikacji produktu 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));
```

Ten przykład powoduje również wydrukowanie pól deskryptora komunikatu przy użyciu klasy MQMD. Metoda copyFrom() klasy com.ibm.mq.headers.MQMD zapełnia obiekt nagłówka z pól deskryptora komunikatu w komunikacie MQMessage, a nie przez odczytywanie 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 za pomocą IBM MQ classes for Java
W tym przykładzie użyto metody indexOf(String) elementu MQHeaderList w celu znalezienia nagłówka MQRFH2 w komunikacie, jeśli taki nagłówek jest obecny.

```
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 produktu IBM MQ classes for Java
W tym przykładzie pokazano, jak można uzyskać dostęp do znanej wartości pola w nazwanym folderze, używając klasy MQRFH2 .

Klasa MQRFH2 udostępnia wiele sposobów uzyskiwania dostępu nie tylko do pól w stałej części struktury, ale także do treści folderu zakodowanego w formacie XML, które są przenoszone w polu NameValueData. W tym przykładzie pokazano, w jaki sposób można uzyskać dostęp do znanej wartości pola w nazwanym folderze-w tym przypadku pole Rto w folderze jms, które reprezentuje nazwę kolejki odpowiedzi w komunikacie MQ JMS .

```
MQRFH2 rfh = ...
```

```
String value = rfh.getStringFieldValue ("jms", "Rto");
```

Aby wykryć treść obiektu MQRFH2 (w przeciwieństwie do bezpośredniego żądania konkretnych pól), można użyć metody getFolders w celu zwrócenia listy MQRFH2.Element, który reprezentuje strukturę folderu, który może zawierać pola i inne foldery. Ustawienie pola lub folderu na wartość NULL powoduje usunięcie go z obiektu MQRFH2. W przypadku 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 używane są klasy nagłówka do analizowania treści nagłówka produktu IBM MQ i manipulowania nimi, gdy źródło danych nie jest obiektem MQMessage.

Klasy nagłówka można używać do analizowania treści nagłówka produktu IBM MQ i manipulowania nimi nawet wtedy, gdy 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 MQMessage, nadpisując kodowanie i identyfikator CCSID określone w deskryptorze komunikatu. Jest to przydatne w przypadku komunikatów, które zawierają łańcuch nagłówków w różnych kodowaniach.

Można również uzyskać obiekty DataInput i DataOutput z innych strumieni danych, na przykład strumienie plików lub gniazd, lub tablice bajtów przenoszone w komunikatach JMS . Klasy java.io.DataInputStream implementują interfejs DataInput , a klasy java.io.DataOutputStream implementują DataOutput. W tym przykładzie odczyta jest treść nagłówka IBM MQ 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 MQHeaderIterator może zostać zastąpiony

```
MQDLH dlh = new MQDLH (in, CMQC.MQENC_NATIVE, CMQC.MQCCSI_DEFAULT);
// or any other header type
```

W tym przykładzie zapis do tablicy bajtów przy użyciu strumienia DataOutputjest następujący:

```
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 dla kodowania i argumentów characterSet . Podczas odczytywania nagłówków należy określić kodowanie i identyfikator CCSID, z którym oryginalnie została zapisana treść bajtowa. Podczas zapisywania nagłówków należy określić kodowanie i identyfikator CCSID, który ma zostać wyproduowany. Konwersja danych jest wykonywana automatycznie przez klasy nagłówka.

Tworzenie klas dla nowych typów nagłówków za pomocą IBM MQ classes for Java
Klasy Java można tworzyć dla typów nagłówków, które nie są dostarczane razem 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 implementuającą interfejs MQHeader. Najprostszym podejściem jest rozszerzenie klasy com.ibm.mq.headers.impl.Header . W tym przykładzie jest tworzony w pełni funkcjonalna klasa reprezentująca strukturę nagłówka MQTM. Nie trzeba dodawać pojedynczych metod pobierających i ustawiających dla każdego pola, ale jest to użyteczna wygodna dla użytkowników klasy nagłówka. Ogólne metody getValue i setValue , które pobierają łańcuch dla nazwy pola, będą działać dla wszystkich pól zdefiniowanych w typie nagłówka. Odziedziczone metody odczytu, zapisu i wielkości umożliwią odczytywanie i zapisywanie instancji nowego typu nagłówka oraz poprawnie obliczą wielkość nagłówka w oparciu o jego definicję pola. Definicja typu jest tworzona tylko raz, jednak tworzone są wiele instancji tej klasy nagłówka. Aby nowa definicja nagłówka była dostępna do dekodowania przy użyciu klas MQHeaderIterator lub MQHeaderList , należy zarejestrować ją za pomocą MQHeaderRegistry. Należy jednak pamiętać, że klasa nagłówka MQTM jest już w rzeczywistości 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 przy użyciu produktu IBM MQ classes for Java

Klasy Java są udostępniane do tworzenia i analizowania komunikatów ustrukturyzowanych PCF, a także do ułatwiania wysyłania żądań PCF i zbierania odpowiedzi PCF.

Klasy PCFMessage & MQCFGR reprezentują tablice struktur parametrów PCF. Udostępniają one podręczne metody dodawania i pobierania parametrów PCF.

Struktury parametrów PCF są reprezentowane przez klasy MQCFH, MQCFIN, MQCFIN64, MQCFST, MQCFBS, MQCFIL, MQCFIL64 MQCFSL i MQCFGR. Te podstawowe interfejsy operacyjne są współużytkowane:

- Metody do odczytu i zapisu treści wiadomości: read (), write () i size ()
- Metody służące do 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 filtrowania. W enkapsulacji w następujących klasach:

- MQCFIF-filtr liczby całkowitej

- MQCFSF-filtr łańcuchów
- MQCFBF-filtr bajtów

Do zarządzania połączeniem z menedżerem kolejek, kolejką serwera komend i powiązaną kolejką odpowiedzi udostępniono dwa klasy agenta, PCFAgent i PCFMessageAgent . Agent PCFMessageAgent rozszerza agent PCFAgent i zwykle powinien być używany do jego preferencji. Klasa PCFMessageAgent przekształca odebrane komunikaty MQMessages i przekazuje je z powrotem do programu wywołującego jako tablicę PCFMessage. Agent PCFAgent zwraca tablicę komunikatów MQMessages, którą 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 produkcie 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 27.

W produkcie IBM MQ classes for Java dostęp do komunikatów odbywa się za pomocą klasy MQMessage. Uchwyty komunikatów nie są więc udostępniane w środowisku Java i nie ma odpowiednika dla funkcji IBM MQ wywołują wywołania MQCRTMH, MQDLTMH, MQMHBUF i MQBUFMH

Aby ustawić właściwości uchwytu komunikatu w interfejsie proceduralnym, należy użyć wywołania MQSETMP. W produkcie IBM MQ classes for Javanależy użyć 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 czasami nazywane wspólnie metodami *set*property* .

Aby zwrócić wartość właściwości uchwytu komunikatu w interfejsie proceduralnym, należy użyć wywołania MQINQMP. W produkcie IBM MQ classes for Javanależy użyć 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 czasami nazywane wspólnie metodami *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

Handle errors arising from IBM MQ classes for Java using Java `try` and `catch` blocks.

Metody w interfejsie Java nie zwracają kodu zakończenia i kodu przyczyny. Zamiast tego zgłaszają one wyjątek, gdy kod zakończenia i kod przyczyny wynikające z wywołania IBM MQ nie są jednocześnie zerami. Upraszcza to logikę programu, tak aby nie trzeba było sprawdzać kodów powrotu po każdym wywołaniu programu IBM MQ. Możesz zdecydować, w jakich punktach w programie chcesz poradzić sobie z możliwością awarii. W tych punktach można surround kodu za pomocą bloków `try` i `catch` , jak w następującym 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 wywołania IBM MQ zgłoszone z powrotem w wyjątkach produktu Java dla produktu z/OS zostały udokumentowane w sekcji Kody zakończenia i przyczyny interfejsu API.

Wyjątki zgłaszane podczas działania aplikacji produktu IBM MQ classes for Java są również zapisywane w dzienniku. Jednak aplikacja może wywołać metodę MQException.logExclude(), aby zapobiec protokołowaniu wyjątków powiązanych z określonym kodem przyczyny. Można to zrobić w sytuacjach, w których oczekiwane jest zgłoszenie wielu wyjątków powiązanych z określonym kodem przyczyny i nie chcesz, aby dziennik był wypełniany tymi wyjątkami. Jeśli na przykład aplikacja próbuje pobrać komunikat z kolejki za każdym razem, gdy iteruje on pętlę, a w przypadku większości tych prób nie oczekuje się, aby w kolejce nie było odpowiedniego komunikatu, można zapobiec zarejestowaniu wyjątków powiązanych z kodem przyczyny MQRC_NO_MSG_AVAILABLE. Jeśli aplikacja wcześniej uniemożliwiła rejestrowanie wyjątków powiązanych z określonym kodem przyczyny, może zezwolić na ponowne zalogowanie się tych wyjątków przez wywołanie metody MQException.logInclude().

Czasami kod przyczyny nie przekazuje wszystkich szczegółów związanych z błędem. W przypadku każdego zgłoszonego wyjątku aplikacja powinna sprawdzić powiązany wyjątek. Sam powiązany wyjątek może mieć inny powiązany wyjątek, a więc połączone wyjątki tworzą łańcuch prowadzący do pierwotnego problemu bazowego. Powiązany wyjątek jest implementowany przy użyciu mechanizmu połączonych wyjątków klasy java.lang.Throwable , a aplikacja uzyskuje powiązany wyjątek, wywołując metodę Throwable.getCause(). Z wyjątku, który jest instancją wyjątku MQException, MQException.getCause() pobiera podstawową 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 dostępne dla wielu wspólnych atrybutów. Dostęp do innych osób można uzyskać za pomocą metod ogólnych inquire () i set ().

W przypadku wielu wspólnych atrybutów klasy MQManagedObject, MQDestination, MQQueue, MQTopic, MQProcess i MQQueueManager zawierają metody getXXX() i setXXX(). Metody te pozwalają na uzyskanie

i ustawienie ich wartości atrybutów. Należy pamiętać, że w przypadku obiektu MQDestination, MQQueue i MQTopic metody działają tylko wtedy, gdy podczas otwierania obiektu określone zostaną odpowiednie opcje zapytania i ustawienia.

W przypadku mniej wspólnych atrybutów klasy MQQueueManager, MQDestination, MQQueue, MQTopic i MQProcess dziedziczą wszystkie klasy z klasy o nazwie MQManagedObject. Ta klasa definiuje interfejsy inquire () i set ().

Po utworzeniu nowego obiektu menedżera kolejek przy użyciu operatora *nowy* jest on automatycznie otwierany dla zapytania. Jeśli do uzyskania dostępu do obiektu procesu używana jest metoda accessProcess(), obiekt ten jest automatycznie otwierany w celu uzyskania informacji. Jeśli do uzyskania dostępu do obiektu kolejki używana jest metoda accessQueue(), obiekt ten nie jest automatycznie otwierany na potrzeby operacji sprawdzania lub ustawiania operacji. Jest to spowodowane tym, że dodanie tych opcji automatycznie może spowodować problemy z niektórymi typami kolejek zdalnych. Aby użyć metod inquire, set, getXXXi setXXX w kolejce, należy określić odpowiednie opcje zapytania i ustawienia w parametrze openOptions metody accessQueue(). To samo dotyczy obiektów docelowych i obiektów tematów.

Metody zapytania i ustawiania przyjmują trzy parametry:

- tablica selektorów
- Tablica intAttrs
- Tablica charAttrs

Nie są potrzebne parametry SelectorCount, IntAttrCount i CharAttrLength, które znajdują się w tabeli MQINQ, ponieważ długość tablicy w Java jest zawsze znana. W poniższym przykładzie przedstawiono sposób wykonania 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 programie Java

Środowisko wykonawcze produktu Java jest z natury wielowątkowe. Produkt IBM MQ classes for Java umożliwia współużytkowanie obiektu menedżera kolejek przez wiele wątków, ale zapewnia, że wszystkie uprawnienia dostępu do docelowego menedżera kolejek są zsynchronizowane.

Programy wielowątkowe są trudne do uniknięcia w programie Java. Rozważmy prosty program, który łączy się z menedżerem kolejek i otwiera kolejkę przy starcie. Program wyświetla na ekranie pojedynczy przycisk. Gdy użytkownik kliknie ten przycisk, program pobierze komunikat z kolejki.

Środowisko wykonawcze produktu Java jest z natury wielowątkowe. W związku z tym inicjowanie aplikacji występuje w jednym wątku, a kod wykonywany w odpowiedzi na naciśnięcie przycisku jest wykonywany w osobnym wątku (wątek interfejsu użytkownika).

W przypadku produktu IBM MQ MQI clientopartego na języku C może to spowodować problem, ponieważ w wielu wątkach istnieją ograniczenia dotyczące współużytkowania uchwytów. Program IBM MQ classes for Java rozluźnia to ograniczenie, umożliwiając współużytkowanie obiektu menedżera kolejek (oraz powiązanej z nim kolejki, tematu i obiektów procesu) przez wiele wątków.

Implementacja produktu IBM MQ classes for Java zapewnia, że dla konkretnego połączenia (instancji obiektuMQQueueManager) wszystkie uprawnienia dostępu do docelowego menedżera kolejek produktu

IBM MQ są synchronizowane. Wątek, który chce wydać wywołanie do 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 współbieżnego dostępu. (Jest to równoznaczne z wywołaniem oddzielnego wywołania MQCONN dla każdego wątku).

Uwaga: Instancje klasy `com.ibm.mq.MQGetMessageOptions` nie mogą być współużytkowane między wątkami, które jednocześnie żądają komunikatów. Instancje tej klasy są aktualizowane przy użyciu danych podczas odpowiadania na odpowiednie żądanie MQGET, co może spowodować nieoczekiwane konsekwencje, gdy wiele wątków działa jednocześnie w tej samej instancji obiektu.

Korzystanie z wyjść kanału w programie IBM MQ classes for Java

Przegląd sposobów korzystania z wyjść kanału w aplikacji za pomocą IBM MQ classes for Java.

W poniższych tematach opisano sposób zapisu wyjścia kanału w produkcie Java, sposobu jego przypisywania oraz sposobu przekazywania danych do niego. Następnie opisują, w jaki sposób używać wyjść kanału napisanych w języku C oraz jak używać sekwencji wyjść kanału.

Aby załadować klasę wyjścia kanału, aplikacja musi mieć poprawne uprawnienia zabezpieczeń.

Tworzenie wyjścia kanału w produkcie IBM MQ classes for Java

Użytkownik może udostępnić własne wyjścia kanału, definiując klasę Java implementuującą odpowiedni interfejs.

Aby zaimplementować wyjście, należy zdefiniować nową klasę Java , która implementuje odpowiedni interfejs. W pakiecie com.ibm.mq.exits są zdefiniowane trzy interfejsy wyjścia:

- WMQSendExit
- WMQReceiveExit
- WMQSecurityExit

Uwaga: Wyjścia kanału są obsługiwane tylko dla połączeń klientów. Nie są one obsługiwane w przypadku połączeń powiązań. Nie można używać wyjścia kanału Java poza programem IBM MQ classes for Java, na przykład w przypadku korzystania z aplikacji klienckiej napisanej w języku C.

Wszystkie szyfrowanie TLS zdefiniowane dla połączenia jest wykonywane *po* wywołaniu funkcji wysyłania i wysyłania zabezpieczeń. Podobnie deszyfrowanie jest wykonywane *przed* wywołaniem odbierania i kończy działanie zabezpieczeń.

Poniższy przykład definiuje 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
```

```
  }
}
```

Każde wyjście jest przekazywane do obiektu MQCXP i do obiektu MQCD. Obiekty te reprezentują struktury MQCXP i MQCD zdefiniowane w interfejsie proceduralnym.

Każda klasa wyjścia, którą należy napisać, musi mieć konstruktor. Może to być konstruktor domyślny lub inny, który pobiera argument łańcuchowy. Jeśli pobiera on łańcuch, dane użytkownika zostaną przekazane do klasy wyjścia podczas jej tworzenia. Jeśli klasa wyjścia zawiera zarówno konstruktor domyślny, jak i konstruktor z jednym argumentem, to pojedynczy konstruktor argumentów ma priorytet.

W przypadku wyjścia 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 .

Najprostszym możliwym organem wyjściowym jest:

```
{ return agentBuffer; }
```

Nie zamykać menedżera kolejek z poziomu wyjścia kanału.

Korzystanie z istniejących klas wyjścia kanału

W wersjach produktu IBM MQ wcześniejszych niż 7.0można 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
Wyjście kanału można przypisać za pomocą programu IBM MQ classes for Java.

W produkcie IBM MQ classes for Javanie ma bezpośredniego odpowiednika dla kanału IBM MQ . Wyjścia kanału są przypisywane do menedżera MQQueueManager. Na przykład, po zdefiniowaniu klasy, która implementuje interfejs WMQSecurityExit , aplikacja może korzystać z wyjścia zabezpieczeń na jeden z czterech sposobów:

- Poprzez przypisanie instancji klasy do pola MQEnvironment.channelSecurityExit przed utworzeniem obiektu MQQueueManager .
- Ustawiając wartość w polu MQEnvironment.channelSecurityExit na łańcuch reprezentujący klasę wyjścia zabezpieczeń przed utworzeniem obiektu MQQueueManager .

- Tworząc parę klucz/wartość we właściwościach hashtable 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 przy użyciu tabeli definicji kanału klienta, musi zostać zapisane z konstruktorem domyślnym. 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 poniższy fragment kodu przedstawia sposób użycia zabezpieczeń, wysyłania i odbierania wyjść implementowanych 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 adres URL tabeli definicji kanału klienta jest przekazywany do menedżera kolejek MQQueueManager, treść definicji kanału klienta określa wyjścia kanału, które mają być używane, a wszystkie definicje wyjścia w środowisku MQEnvironment lub właściwości hashtable właściwości są ignorowane.
2. Jeśli nie jest przekazywany adres URL CCDT, scalane są 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 7.0, oraz pole channelSendExit, które może być używane dla dowolnego wyjścia wysyłania), to zamiast starego wyjścia jest używane nowe wyjście (channelSendExit).

Jeśli zadeklarowano wyjście kanału jako łańcuch, należy włączyć program IBM MQ w celu zlokalizowania programu obsługi wyjścia kanału. Można to zrobić na różne sposoby, w zależności od środowiska, w którym aplikacja jest uruchomiona, oraz w jaki sposób programy obsługi wyjścia kanału są spakowane.

- W przypadku aplikacji działającej na serwerze aplikacji należy zapisać pliki w katalogu wyświetkowanym w produkcie Tabela 57 na stronie 390 lub spakowane w plikach JAR, do których odwołuje się produkt **exitClasspath**.
- W przypadku aplikacji, która nie jest uruchomiona na serwerze aplikacji, mają zastosowanie następujące reguły:
  - Jeśli klasy wyjścia kanału są spakowane w oddzielnych plikach JAR, te pliki JAR muszą zostać dołączone do produktu **exitClasspath**.
  - Jeśli klasy wyjścia kanału nie są spakowane w plikach JAR, pliki klas mogą być przechowywane w katalogu podanym w katalogu Tabela 57 na stronie 390 lub w dowolnym katalogu w ścieżce klasy systemowej maszyny JVM lub w katalogu **exitClasspath**.

Właściwość **exitClasspath** może być określona 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 w pliku `mqclient.ini`
3. Pozycja tabeli mieszającej z kluczem CMQC.EXIT_CLASSPATH_PROPERTY

4. Zmienna MQEnvironment **exitClasspath**

Wiele ścieżek należy rozdzielić za pomocą znaku java.io.File.pathSeparator .

Tabela 57. Katalog dla programów obsługi wyjścia kanału	
Platforma	**Katalog**
 AIX  Linux  Solaris	`/var/mqm/exits` (32-bitowe programy obsługi wyjścia kanału) `/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 programu 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

W przypadku wyjścia wysyłania parametr *agentBuffer* zawiera dane, które mają zostać wysłane. W przypadku wyjścia odbierania lub wyjścia zabezpieczeń parametr *agentBuffer* zawiera dane, które właśnie zostały odebrane. Parametr length nie jest wymagany, ponieważ wyrażenie agentBuffer.limit () wskazuje długość tablicy.

W przypadku wyjścia 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 .

Najprostszym możliwym organem wyjściowym jest:

```
{ return agentBuffer; }
```

Wyjścia kanału są wywoływane z buforem, w którym znajduje się tablica zapasowa. Aby uzyskać najlepszą wydajność, wyjście powinno zwrócić bufor z tablicą bazową.

Dane użytkownika

Jeśli aplikacja łączy się z menedżerem kolejek, ustawiając opcję channelSecurityExit, channelSendExit lub channelReceiveExit, 32 bajty danych użytkownika mogą być przekazywane do odpowiedniej klasy wyjścia kanału, gdy jest wywoływana, za pomocą pól channelSecurityExitUserData, channelSendExitUserData lub channelReceiveExitUserData. Te dane użytkownika są dostępne dla klasy wyjścia kanału, ale są odświeżane za każdym razem, gdy wywoływane jest wyjście. W związku z tym wszelkie zmiany wprowadzone w danych użytkownika w wyjściu kanału zostaną utracone. Aby trwałe zmiany danych w wyjściu kanału były wprowadzane, należy użyć obszaru exitUserproduktu MQCXP. Dane w tym polu są przechowywane między wywołaniami wyjścia.

Jeśli aplikacja ustawi wartości securityExit, sendExitlub receiveExit, dane użytkownika nie mogą być przekazywane do tych klas wyjścia kanału.

Jeśli aplikacja korzysta z tabeli definicji kanału klienta (CCDT) w celu 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 korzystania z tabeli definicji kanału klienta zawiera sekcja “Korzystanie z tabeli definicji kanału klienta przy użyciu produktu IBM MQ classes for Java” na stronie 372.

Używanie wyjść kanału, które nie zostały zapisane w programie Java z programem IBM MQ classes for Java
W jaki sposób używać programów obsługi wyjścia kanału napisanych w języku C z aplikacji Java .

W programie IBM WebSphere MQ 7.0można określić nazwę programu obsługi wyjścia kanału napisanego w języku C jako łańcuch przekazany do pól wyjściowych channelSecurityExit, channelSendExit lub channelReceivew obiekcie MQEnvironment lub w tabeli Hashtable. Nie można jednak użyć wyjścia kanału napisanego w programie Java w aplikacji napisanej w innym języku.

Podaj nazwę programu obsługi wyjścia w formacie `library(function)` i upewnij się, że położenie programu obsługi wyjścia zostało określone zgodnie z opisem w sekcji Ścieżka do wyjść.

Więcej informacji na temat pisania wyjścia kanału w języku C zawiera sekcja“Programy obsługi wyjścia kanału dla kanałów przesyłania komunikatów” na stronie 1059.

Korzystanie z zewnętrznych klas wyjścia

W wersjach wcześniejszych niż IBM WebSphere MQ 7.0udostępniono trzy klasy, które umożliwiały korzystanie z wyjść kanału napisanych w językach innych niż Java:

- MQExternalSecurityWyjście, które implementuje interfejs MQSecurityExit
- Wyjście MQExternalSend, które implementuje interfejs MQSendExit
- Wyjście MQExternalReceive, które implementuje interfejs MQReceiveExit

Korzystanie z tych klas pozostaje poprawne, ale preferowana jest nowa metoda.

Aby użyć wyjścia zabezpieczeń, które nie zostało zapisane w produkcie Java, aplikacja najpierw musiała utworzyć obiekt wyjścia MQExternalSecurity. Podana aplikacja, jako parametry w konstruktorze wyjścia MQExternalSecurity, nazwa biblioteki zawierającej wyjście zabezpieczeń, nazwa punktu wejścia dla wyjścia zabezpieczeń oraz dane użytkownika, które mają być przekazane do wyjścia zabezpieczeń podczas jego wywołania. Programy obsługi wyjścia kanału, które nie są zapisywane w programie Java , zostały zapisane w katalogu podanym w sekcjiTabela 57 na stronie 390.

Korzystanie z sekwencji wyjść wysyłania lub odbierania kanału w programie IBM MQ classes for Java
Aplikacja IBM MQ classes for Java może korzystać z sekwencji wyjść wysyłania lub odbierania kanału, które są uruchamiane w ramach dziedziczenia.

Aby użyć sekwencji wysyłania wyjść, 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 (w przypadku wyjścia wysyłania napisanego w produkcie Java)
- Instancja klasy wyjścia MQExternalSend(w przypadku wyjścia wysyłania nie napisanego w produkcie Java)
- Instancja klasy łańcucha MQSendExit
- Instancja klasy String

Lista nie może zawierać innej listy.

Aplikacja może korzystać z sekwencji wyjść odbierania w podobny sposób.

Jeśli używany jest łańcuch, musi on składać się z jednej lub większej liczby definicji wyjścia oddzielonych przecinkami, z których każda może być nazwą klasy Java lub programu w formacie `library(function)`.

Następnie aplikacja przypisuje obiekt List lub String do pola MQEnvironment.channelSendExit przed utworzeniem obiektu MQQueueManager .

Kontekst informacji przekazywanych do wyjść jest wyłącznie w obrębie domeny wyjść. Na przykład, jeśli wyjście Java i wyjście C są połączone w łańcuchy, obecność wyjścia Java nie ma wpływu na wyjście C.

Korzystanie z klas łańcucha wyjścia

W wersjach wcześniejszych niż IBM WebSphere MQ 7.0udostępniono dwie klasy umożliwiające sekwencje wyjść:

- MQSendExitŁańcuch, który implementuje interfejs MQSendExit
- MQReceiveExitŁańcuch, który implementuje interfejs MQReceiveExit

Korzystanie z tych klas pozostaje poprawne, ale preferowana jest nowa metoda. Użycie interfejsów IBM MQ Klasy dla Java oznacza, że aplikacja nadal ma zależność od `com.ibm.mq.jar` Jeśli nowy zestaw interfejsów w pakiecie `com.ibm.mq.exits` jest używany, nie ma zależności od `com.ibm.mq.jar`.

Aby użyć sekwencji wysyłania wyjść, aplikacja utworzyła listę obiektów, w której każdy obiekt był jednym z następujących obiektów:

- Instancja klasy zdefiniowanej przez użytkownika, która implementuje interfejs MQSendExit (w przypadku wyjścia wysyłania napisanego w produkcie Java)
- Instancja klasy wyjścia MQExternalSend(w przypadku wyjścia wysyłania nie napisanego w produkcie Java)
- Instancja klasy łańcucha MQSendExit

Aplikacja utworzyła obiekt łańcucha MQSendExit, przekazując tę listę obiektów jako parametr w konstruktorze. Aplikacja przypisałaby następnie obiekt MQSendExitdo pola MQEnvironment.sendExit przed utworzeniem obiektu MQQueueManager .

Kompresja kanału w produkcie IBM MQ classes for Java

Kompresowanie danych, które przepływa na kanał, może poprawić wydajność kanału i zmniejszyć ruch w sieci. Produkt IBM MQ classes for Java używa funkcji kompresji wbudowanej w produkt IBM MQ.

Za pomocą funkcji dostarczonej z produktem IBM MQmożna kompresować dane, które przepływają w kanałach komunikatów i kanałach MQI, a także-w obu typach kanałów-można kompresować dane nagłówka i dane komunikatów niezależnie od siebie. Domyślnie żadne dane nie są kompresowane w kanale. Pełny opis kompresji kanałów, w tym sposób jego implementacji w produkcie IBM MQ, zawiera sekcja Kompresja danych (COMPMSG) i Kompresja nagłówka (COMPHDR).

Aplikacja IBM MQ classes for Java określa techniki, które mogą być używane do kompresowania nagłówka lub danych komunikatów 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, jest to kolejność, w jakiej techniki kompresji są negocjowane z menedżerem kolejek po uruchomieniu połączenia klienckiego. Aplikacja może następnie przypisać kolekcję do pola hdrCompList, dla danych nagłówka lub pola msgCompList, dla danych komunikatu, w klasie MQEnvironment. Gdy aplikacja jest gotowa, może ona uruchomić połączenie klienta, tworząc obiekt MQQueueManager .

Opisane poniżej fragmenty kodu ilustrują opisane podejście. Pierwszy fragment kodu pokazuje, jak zaimplementować kompresję 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 rozpoczyna się połączenie z klientem. Wybrana technika kompresji nie może zostać zmieniona w czasie życia obiektu MQQueueManager .

Techniki kompresji dla danych nagłówka i komunikatu, które są obsługiwane zarówno przez klienta, jak i menedżera kolejek w połączeniu klienta, są przekazywane do wyjścia kanału jako kolekcje w polach listy hdrCompi msgCompw obiekcie MQChannelDefinition . Rzeczywiste techniki, które są obecnie używane do kompresowania nagłówka i danych komunikatów w połączeniu klienta, są przekazywane do wyjścia kanału w polach kompresji CurHdri kompresji CurMsgobiektu MQChannelExit .

Jeśli kompresja jest używana w połączeniu z klientem, dane są kompresowane przed przetworami i wyodrębnionymi wyjściami nadawczego kanału po przetworzeniu wszystkich wyjść odbierania kanału. Dane przekazywane do wyjścia wysyłania i odbierania są w związku z tym w stanie skompresowanym.

Więcej informacji na temat określania technik kompresji i technik kompresji można znaleźć w sekcji Klasa com.ibm.mq.MQEnvironment i Interfejs com.ibm.mq.MQC.

Współużytkowanie połączenia TCP/IP w produkcie IBM MQ classes for Java

W celu współużytkowania pojedynczego połączenia TCP/IP można utworzyć wiele instancji kanału MQI.

W produkcie IBM MQ classes for Javaużywana jest zmienna MQEnvironment.sharingConversations , która umożliwia sterowanie liczbą konwersacji, które mogą współużytkować pojedyncze połączenie TCP/IP.

Atrybut SHARECNV jest najlepszym sposobem podejścia do współużytkowania połączeń. W związku z tym, gdy wartość SHARECNV większa niż 0 jest używana razem z IBM MQ classes for Java , nie ma gwarancji, że nowe żądanie połączenia zawsze będzie współużytkować już nawiązane połączenie.

Zestawianie połączeń w produkcie IBM MQ classes for Java

Produkt IBM MQ classes for Java umożliwia łączenie zapasowych połączeń w celu ich ponownego wykorzystania.

Produkt IBM MQ classes for Java udostępnia dodatkowe wsparcie dla aplikacji, które zajmują się wieloma połączeniami z menedżerami kolejek produktu IBM MQ . Gdy połączenie nie jest już potrzebne, zamiast niszczyć go, można je połączyć i później ponownie wykorzystać. Może to zapewnić znaczące zwiększenie wydajności aplikacji i oprogramowania pośredniego, które łączą się szeregowo z dowolnymi menedżerami kolejek.

Produkt IBM MQ udostępnia domyślną pulę połączeń. Aplikacje mogą aktywować lub dezaktywować tę pulę połączeń poprzez zarejestrowanie i wyrejestrowywanie tokenów za pomocą klasy MQEnvironment. Jeśli pula jest aktywna, gdy program IBM MQ classes for Java konstruuje obiekt MQQueueManager , przeszukuje tę pulę domyślną i ponownie użyje odpowiedniego połączenia. Gdy wystąpi wywołanie MQQueueManager.disconnect (), bazowe połączenie jest zwracane do puli.

Alternatywnie aplikacje mogą skonstruować pulę połączeń menedżera MQSimpleConnectiondla konkretnego zastosowania. Następnie aplikacja może określić tę pulę podczas budowy obiektu MQQueueManager lub przekazać tę pulę do środowiska MQEnvironment w celu użycia jako domyślnej puli połączeń.

Aby zapobiec korzystaniu z zbyt dużej ilości zasobów, można ograniczyć łączną liczbę połączeń, które może obsłużyć obiekt MQSimpleConnectionManager, a także ograniczyć wielkość puli połączeń. Ustawienie limitów jest przydatne, jeśli występują konflikty żądań połączeń w obrębie maszyny JVM.

Domyślnie metoda getMaxConnections () zwraca wartość zero, co oznacza, że nie ma limitu liczby połączeń, które mogą być obsługiwane przez obiekt MQSimpleConnectionManager. Limit można ustawić za pomocą metody setMaxConnections (). Jeśli zostanie ustawiony limit, 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 korzystania z domyślnej puli połączeń.

Należy rozważyć zastosowanie następującej przykładowej aplikacji 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();
         }
      }
}
```

Program MQApp1 pobiera z wiersza komend listę lokalnych menedżerów kolejek, a następnie łączy się z każdym z kolei i wykonuje pewne operacje. Jednak w przypadku, gdy wiersz komend zawiera wiele razy ten sam menedżer kolejek, można połączyć się z tym samym menedżerem kolejek tylko raz, a także wielokrotnie ponownie wykorzystywać to połączenie.

Produkt IBM MQ classes for Java udostępnia domyślną pulę połączeń, która może być używana do wykonania tej czynności. Aby włączyć tę pulę, należy użyć jednej z metod MQEnvironment.addConnectionPoolToken(). Aby wyłączyć pulę, należy użyć metody MQEnvironment.removeConnectionPoolToken().

Następująca przykładowa aplikacja MQApp2jest 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 przez zarejestrowanie obiektu 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 qmgr.disconnect() zwraca połączenie z pulą w celu późniejszego ponownego wykorzystania. Te wywołania funkcji API są takie same, jak przykładowa aplikacja MQApp1.

Druga podświetlona linia dezaktywuje domyślną pulę połączeń, która niszczy wszystkie połączenia menedżera kolejek zapisane w puli. Jest to ważne, ponieważ w przeciwnym razie aplikacja zostanie zakończona z pewną liczbą aktywnych połączeń menedżera kolejek w puli. Ta sytuacja może powodować błędy, które pojawiają się w dziennikach menedżera kolejek.

Jeśli aplikacja korzysta z tabeli definicji kanału klienta (CCDT) w celu nawiązania połączenia z menedżerem kolejek, konstruktor MQQueueManager najpierw wyszukuje odpowiednią definicję kanału połączenia klienta. Jeśli zostanie znaleziony, konstruktor przeszukuje domyślną pulę połączeń dla połączenia, które może być użyte dla kanału. Jeśli konstruktor nie może znaleźć odpowiedniego połączenia w puli, przeszukuje tabelę definicji kanału klienta dla następnej odpowiedniej definicji kanału

połączenia klienta i kontynuuje działanie zgodnie z opisem poprzednio. Jeśli konstruktor zakończy wyszukiwanie w tabeli definicji kanału klienta i nie znajdzie żadnego odpowiedniego połączenia w puli, konstruktor rozpocznie drugie wyszukiwanie w tabeli. Podczas tego wyszukiwania konstruktor próbuje utworzyć nowe połączenie dla każdej odpowiedniej definicji kanału połączenia klienckiego, a następnie użyje pierwszego połączenia, które ma zostać utworzone.

Domyślna pula połączeń przechowuje maksymalnie dziesięć nieużywanych połączeń i utrzymuje nieużywane połączenia aktywne przez maksymalnie pięć minut. Aplikacja może to zmienić (szczegółowe informacje można znaleźć w sekcji “Dostarczanie innej puli połączeń w produkcie IBM MQ classes for Java” na stronie 396).

Zamiast używać środowiska MQEnvironment do dostarczania znacznika MQPoolToken, aplikacja może utworzyć własne elementy:

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

Niektóre aplikacje lub dostawcy oprogramowania pośredniego udostępniają podklasy elementu MQPoolToken w celu przekazania informacji do niestandardowej puli połączeń. Można je utworzyć i przekazać do metody addConnectionPoolToken() w ten sposób, aby dodatkowe informacje mogły zostać przekazane do puli połączeń.

Domyślna pula połączeń i wiele komponentów w produkcie IBM MQ classes for Java
W tym przykładzie pokazano, jak dodać lub usunąć element MQPoolTokens ze statycznego zestawu zarejestrowanych obiektów MQPoolToken .

Środowisko MQEnvironment zawiera statyczny zestaw zarejestrowanych obiektów MQPoolToken . Aby dodać lub usunąć element 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 element MQPoolToken do zestawu MQEnvironment ustawionego na jego czas życia.

Na przykład przykładowa aplikacja MQApp3 tworzy dziesięć wątków i uruchamia każdą z nich. Każdy wątek rejestruje swój własny obiekt MQPoolToken, czeka przez pewien czas, a następnie łączy się z menedżerem kolejek. Po rozłączonym wątku usuwa on własny element MQPoolToken.

Domyślna pula połączeń pozostaje aktywna, gdy w zestawie MQPoolTokensznajduje się co najmniej jeden znacznik, więc pozostanie on aktywny przez czas trwania tej aplikacji. Aplikacja nie musi przechowywać obiektu głównego w ogólnej kontroli wątków.

```
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);
      }
}
```

Dostarczanie innej puli połączeń w produkcie IBM MQ classes for Java
W tym przykładzie przedstawiono sposób użycia klasy **com.ibm.mq.MQSimpleConnectionManager** w celu dostarczenia innej puli połączeń.

Ta klasa udostępnia podstawowe narzędzia do zestawiania połączeń, a aplikacje mogą korzystać z tej klasy w celu dostosowania zachowania puli.

Po utworzeniu instancji menedżer MQSimpleConnectionmoże zostać określony w konstruktorze MQQueueManager . Menedżer MQSimpleConnectionzarządza następnie połączeniem, który stanowi podstawę dla skonstruowanego menedżera MQQueueManager. Jeśli menedżer MQSimpleConnectionzawiera odpowiednie połączenie z puli, to połączenie jest 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 zapisywane w pliku myConnMan po wywołaniu metody qmgr.disconnect(). Połączenie jest następnie ponownie wykorzystywane podczas drugiego wywołania konstruktora MQQueueManager .

Drugi wiersz umożliwia włączenie menedżera MQSimpleConnection. Ostatni wiersz wyłącza menedżer MQSimpleConnection, niszcząc wszystkie połączenia znajdujące się w puli. Menedżer MQSimpleConnectionjest domyślnie w trybie MODE_AUTO, który jest opisany w dalszej części tej sekcji.

Menedżer MQSimpleConnectionumożliwia przydzielaniu połączeń na podstawie ostatnio używanej bazy danych i niszczy połączenia w najmniejszym-ostatnio używanym środowisku. Domyślnie połączenie jest niszczone, jeśli nie zostało użyte przez pięć minut, lub jeśli w puli znajduje się więcej niż dziesięć nieużywanych połączeń. Wartości te można zmienić, wywołując metodę MQSimpleConnectionManager.setTimeout().

Można również skonfigurować menedżer MQSimpleConnection, który będzie używany jako domyślna pula połączeń, jeśli menedżer połączeń nie jest dostępny w konstruktorze MQQueueManager .

Poniższa aplikacja demonstruje, że:

```
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 wiersze tworzą i konfigurują obiekt menedżera MQSimpleConnection. Konfiguracja wykonuje następujące czynności:

- Kończy połączenia, które nie są używane przez godzinę
- Ogranicza liczbę połączeń zarządzanych przez myConnMan do 75
- Ogranicza liczbę nieużywanych połączeń w puli do 50.
- Ustawia wartość MODE_AUTO, która jest wartością domyślną. Oznacza to, że pula jest aktywna tylko wtedy, gdy jest domyślnym menedżerem połączeń i istnieje co najmniej jeden znacznik w zestawie MQPoolTokens , który jest organizowany przez produkt MQEnvironment.

Nowy menedżer MQSimpleConnectionjest następnie ustawiany jako domyślny menedżer połączeń.

W ostatnim wierszu aplikacja wywołuje komendę MQApp3.main(). Spowoduje to uruchomienie pewnej liczby wątków, w których każdy wątek używa produktu IBM MQ niezależnie. Te wątki używają programu myConnMan, gdy są one wymuszane połączenia.

Koordynacja JTA/JDBC przy użyciu produktu IBM MQ classes for Java

Produkt IBM MQ classes for Java obsługuje metodę MQQueueManager.begin (), która umożliwia IBM MQ działanie jako koordynator dla bazy danych, która udostępnia sterownik JDBC typu 2 lub JDBC typu 4 zgodny ze standardem.

Ta obsługa nie jest dostępna na wszystkich platformach. Aby sprawdzić, które platformy obsługują koordynację JDBC , należy zapoznać się z Wymagania systemowe produktu IBM MQ.

Aby korzystać z obsługi XA-JTA, należy skorzystać ze specjalnej biblioteki przełącznika JTA. Metoda korzystania z tej biblioteki różni się w zależności od tego, czy używany jest produkt Windows , czy też jeden z innych platform.

Konfigurowanie koordynacji JTA/JDBC w systemie Windows

Biblioteka XA jest dostarczana jako biblioteka DLL o nazwie formatu `jdbcxxx.dll`.

Dostarczona produkt `jdbcora12.dll` zapewnia kompatybilność z bazą danych Oracle 12Cw przypadku instalacji serwera IBM MQ dla serwera Windows .

W systemach Windows biblioteka XA jest dostarczana jako kompletna 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 bibliotekę XA, która jest również opisana jako plik ładowania przełącznika, do menedżera kolejek. Należy używać komponentu IBM MQ Explorer. Określ szczegóły pliku ładowania przełącznika na panelu właściwości menedżera kolejek pod kontrolą menedżera zasobów XA. Należy podać tylko nazwę biblioteki. Na przykład:

W przypadku bazy danych Db2 ustaw pole SwitchFile na wartość: dbcdb2

W przypadku bazy danych Oracle ustaw pole SwitchFile na wartość: `jdbcora`

Uwagi:

1. Baza danych Oracle 12C jest obsługiwana przez serwer IBM MQ classes for Java, tylko w systemie IBM MQ dla produktu Windows.
2. Obsługiwana wersja produktu Oracle 12C to 12.1.0.1.0 Enterprise Edition i przyszłe pakiety poprawek.
3. 64-bitowe bazy danych Oracle w 64-bitowym systemie Windows wymagają 32-bitowego klienta Oracle .

4. Korzystając z IBM MQ classes for Java, IBM MQ może pełnić rolę koordynatora transakcji. Nie jest jednak możliwe uczestniczenie w transakcji typu JTA.

Konfigurowanie koordynacji JTA/JDBC na platformach innych niż Windows
Pliki obiektów są dostarczane. Powiąż odpowiedni plik z podanym plikiem makefile i zadeklaruj go w menedżerze kolejek przy użyciu pliku konfiguracyjnego.

Dla każdego systemu zarządzania bazami danych program IBM MQ udostępnia dwa pliki obiektów. Aby utworzyć 32-bitową bibliotekę przełącznika, należy utworzyć dowiązanie do jednego pliku obiektu, a następnie utworzyć dowiązanie do innego pliku wynikowego, aby utworzyć 64-bitową bibliotekę przełącznika. W przypadku bazy danych Db2nazwa każdego pliku wynikowego to jdbcdb2.o , a w przypadku bazy danych Oraclenazwą każdego pliku obiektu jest jdbcora.o.

Każdy plik obiektu musi być dowiązany za pomocą odpowiedniego pliku makefile dostarczanego razem z produktem IBM MQ. Biblioteka przełączników wymaga innych bibliotek, które mogą być przechowywane w różnych położeniach w różnych systemach. Biblioteka przełącznika nie może jednak użyć zmiennej środowiskowej ścieżki biblioteki do zlokalizowania tych bibliotek, ponieważ biblioteka przełączników jest ładowana przez menedżer kolejek, który działa w środowisku setuid. Podany plik makefile zapewnia więc, że biblioteka przełączników zawiera pełne nazwy ścieżek dla tych bibliotek.

Aby utworzyć bibliotekę przełącznika, należy wprowadzić komendę **make** z następującym formatem. Aby utworzyć 32-bitową bibliotekę przełącznika, należy wprowadzić 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* jest systemem zarządzania bazami danych, dla którego tworzona jest biblioteka przełącznika. The valid values are db2 for Db2 and `oracle` for Oracle.

Poniżej przedstawiono przykład komendy **make** :

```
make db2
```

Należy zwrócić uwagę na następujące kwestie:

- Aby uruchomić 32-bitowe aplikacje, należy utworzyć zarówno 32-bitową, jak i 64-bitową bibliotekę przełączników dla każdego używanego systemu zarządzania bazami danych. Aby uruchomić aplikacje 64-bitowe, należy utworzyć tylko 64-bitową bibliotekę przełączników. W przypadku bazy danych Db2nazwą każdej biblioteki przełącznika jest jdbcdb2 , a w przypadku bazy danych Oraclenazwą każdej biblioteki przełącznika jest jdbcora. Pliki makefile zapewniają, że 32-bitowe i 64-bitowe biblioteki przełączników są przechowywane w różnych katalogach IBM MQ . 32-bitowa biblioteka przełączników 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 w celu znalezienia miejsca, w którym zainstalowano bazę danych Oracle .

Po utworzeniu bibliotek przełącznika dla produktu Db2, Oraclelub obu tych bibliotek należy je zadeklarować do menedżera 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 za pomocą jednej z następujących sekcji:

Dla bazy danych Db2

```
SwitchFile=jdbcdb2
```

Dla bazy danych Oracle

```
SwitchFile=jdbcora
```

Nie należy podawać pełnej nazwy ścieżki dla 32-bitowej lub 64-bitowej biblioteki przełącznika. Należy podać tylko nazwę biblioteki.

Jeśli plik konfiguracyjny menedżera kolejek nie zawiera już sekcji XAResourceManager dla baz danych Db2 lub Oracle lub jeśli chcesz dodać dodatkowe sekcje XAResourceManager , zapoznaj się z sekcją Administrowanie , aby uzyskać informacje na temat tworzenia sekcji XAResourceManager . Jednak każda pozycja SwitchFile w nowej sekcji XAResourceManager musi być dokładnie taka jak opisana wcześniej w przypadku bazy danych Db2 lub Oracle . Należy również dołączyć wpis `ThreadOfControl=PROCESS`.

Po zaktualizowaniu pliku konfiguracyjnego menedżera kolejek i upewniu się, że zostały ustawione wszystkie odpowiednie zmienne środowiskowe bazy danych, można zrestartować menedżer kolejek.

Korzystanie z koordynacji JTA/JDBC
Należy zakodować wywołania funkcji 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 metody getJDBCConnection jest implementacją interfejsu XADataSource specyficzną dla bazy danych, która definiuje szczegóły bazy danych, z którą ma zostać nawiązane połączenie. Aby określić, w jaki sposób utworzyć odpowiedni obiekt XADataSource , który ma zostać przekazany do funkcji getJDBCConnection, należy zapoznać się z dokumentacją bazy danych.

Należy również zaktualizować ścieżkę klasy z odpowiednimi plikami jar specyficznymi dla bazy danych w celu wykonania pracy JDBC .

Jeśli konieczne jest nawiązanie połączenia z wieloma bazami danych, należy kilkakrotnie wywołać funkcję getJDBCConnection , aby wykonać transakcję na kilku różnych połączeniach.

Istnieją dwie formy metody getJDBCConnection, które są odzwierciedleniem dwóch form 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
```

Metody te deklarują wyjątek w klauzulach throws w celu uniknięcia problemów z weryfikatorem maszyny JVM dla klientów, którzy nie korzystają z funkcji JTA. Rzeczywisty zgłoszony wyjątek to javax.transaction.xa.XAException , który wymaga dodania pliku jta.jar do ścieżki klasy dla programów, które wcześniej nie wymagali tego pliku.

Aby korzystać z obsługi JTA/JDBC , w aplikacji należy umieścić następującą instrukcję:

```
MQEnvironment.properties.put(CMQC.THREAD_AFFINITY_PROPERTY, new Boolean(true));
```

Znane problemy i ograniczenia związane z koordynacją JTA/JDBC
Niektóre problemy i ograniczenia obsługi JTA/JDBC zależą od używanego systemu zarządzania bazami danych, na przykład przetestowane sterowniki JDBC zachowują się inaczej, gdy baza danych jest

wyłączona, gdy aplikacja jest uruchomiona. Jeśli połączenie z bazą danych, z której korzysta aplikacja, jest zerwane, istnieją kroki, które aplikacja może wykonać w celu ponownego nawiązania nowego połączenia z menedżerem kolejek i bazą danych, tak aby możliwe było użycie tych nowych połączeń w celu wykonania wymaganych prac transakcyjnych.

Ze względu na to, że obsługa JTA/JDBC wywołuje sterowniki JDBC , implementacja tych sterowników JDBC może mieć istotny wpływ na zachowanie systemu. W szczególności testowane sterowniki JDBC zachowują się inaczej, gdy baza danych jest wyłączona, gdy aplikacja jest uruchomiona.

Ważne: Zawsze należy unikać nagłego zamykania bazy danych, gdy istnieją aplikacje, które mają do niego otwarte połączenia.

Uwaga: Aby program IBM MQ działał jako koordynator bazy danych, aplikacja IBM MQ classes for Java musi łączyć się za pomocą trybu powiązań.

Wiele sekcji XAResourceManager
Korzystanie z więcej niż jednej sekcji XAResourceManager w pliku konfiguracyjnym menedżera kolejek `qm.ini`nie jest obsługiwane. Każda sekcja XAResourceManager inna niż ta 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 można znaleźć w dokumentacji produktu Db2 .

Sekcja XAResourceManager musi być skonfigurowana tak, aby używana była opcja ThreadOfControl=PROCESS. W przypadku systemu Db2 8.1 i wyższego nie jest to zgodne z domyślnym wątkiem ustawienia elementu sterującego dla Db2, dlatego wartość toc=p musi być określona w łańcuchu Open String XA. Przykładowa sekcja XAResourceManager dla produktu Db2 z koordynacją JTA/JDBC jest następująca:

```
XAResourceManager:
     Name=jdbcdb2
     SwitchFile=jdbcdb2
     XAOpenString=uid=userid,db=dbalias,pwd=password,toc=p
     ThreadOfControl=PROCESS
```

Nie stanowi to przeszkody dla aplikacji Java , które używają koordynacji JTA/JDBC do obsługi wielowątkowej.

Oracle
Wywołanie metody JDBC Connection.close() po wywołaniu metody MQQueueManager.disconnect () powoduje wygenerowanie wyjątku SQLException. Należy wywołać metodę Connection.close() przed wywołaniem metody MQQueueManager.disconnect () lub pominąć wywołanie metody Connection.close().

Obsługa problemów z połączeniami z bazą danych

Gdy aplikacja IBM MQ classes for Java korzysta z obsługi JTA/JDBC udostępnianej przez produkt IBM MQ, zwykle wykonuje następujące kroki:

1. Tworzy nowy obiekt MQQueueManager w celu reprezentowania połączenia z menedżerem kolejek, który będzie działał jako menedżer transakcji.
2. Tworzy obiekt XADataSource , który zawiera szczegółowe informacje na temat sposobu łączenia się z bazą danych, która zostanie wymieniona w transakcji.
3. Wywołuje metodę MQQueueManager.getJDBCConnection(XADataSource) przechodząca w XADataSource , która została wcześniej utworzona. Powoduje to, że IBM MQ classes for Java nawiąże połączenie z bazą danych.
4. Wywołuje metodę MQQueueManager.begin (), aby uruchomić transakcję XA.

5. Wykonuje pracę przesyłania komunikatów i bazy danych.
6. Po zakończeniu wszystkich wymaganych prac wywołaj metodę 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 działania aplikacji powinna zamknąć połączenie z bazą danych, które zostało utworzone w kroku 3, a następnie wywołać metodę MQQueueManager.disconnect (), aby rozłączyć się z menedżerem kolejek.

IBM MQ classes for Java obsługuje wewnętrzną listę wszystkich połączeń z bazą danych, które zostały utworzone, gdy aplikacja wywołuje komendę MQQueueManager.getJDBCConnection(XADataSource). Jeśli menedżer kolejek musi komunikować się z bazą danych podczas przetwarzania transakcji XA, odbywa się następujące przetwarzanie:

1. Menedżer kolejek wywołuje IBM MQ classes for Java, przekazując szczegóły wywołania XA, które musi zostać przekazane do bazy danych.
2. Następnie IBM MQ classes for Java należy sprawdzić odpowiednie połączenie na liście, a następnie użyć tego połączenia do przepływu wywołania XA do bazy danych.

Jeśli połączenie z bazą danych zostanie utracone w dowolnym momencie tego przetwarzania, aplikacja powinna:

1. Wycofaj wszystkie istniejące prace, które zostały wykonane w ramach transakcji, wywołując metodę MQQueueManager.backout ().
2. Zamknij połączenie z bazą danych. Powinno to spowodować, że program IBM MQ classes for Java usunie szczegóły zerwania połączenia z bazą danych z wewnętrznej listy.
3. Odłącz się od menedżera kolejek, wywołując metodę MQQueueManager.disconnect ().
4. Utwórz 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.

Dzięki temu aplikacja może ponownie nawiązać nowe połączenie z menedżerem kolejek i bazą danych, a następnie użyć tych połączeń w celu wykonania wymaganych prac transakcyjnych.

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 wymagane jest użycie dostawcy JSSE.

Aplikacje klienckie produktu 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 dowolnymi dwoma równorzędnymi płatnikami w sieci Internet lub w intranecie.

Produkt IBM MQ classes for Java używa produktu Java Secure Socket Extension (JSSE) do obsługi szyfrowania TLS, dlatego wymaga dostawcy JSSE. Maszyny JVM JSE v1.4 mają wbudowaną dostawcę JSSE. Szczegółowe informacje na temat zarządzania certyfikatami i ich przechowywania mogą być różne od dostawcy. Aby uzyskać informacje na ten temat, zapoznaj się z dokumentacją dostawcy JSSE.

W tej sekcji założono, że dostawca JSSE jest poprawnie zainstalowany i skonfigurowany oraz że odpowiednie certyfikaty zostały zainstalowane i udostępnione dla dostawcy JSSE.

Jeśli aplikacja kliencka IBM MQ classes for Java korzysta z tabeli definicji kanału klienta (CCDT) do łączenia się z menedżerem kolejek, należy zapoznać się z "Korzystanie z tabeli definicji kanału klienta przy użyciu produktu IBM MQ classes for Java" na stronie 372.

Włączanie protokołu TLS w produkcie IBM MQ classes for Java

Aby włączyć protokół TLS, należy określić pakiet CipherSuite. Istnieją dwa sposoby określania pakietu CipherSuite.

Protokół TLS jest obsługiwany tylko dla połączeń klienckich. Aby włączyć obsługę protokołu TLS, należy określić parametr CipherSuite , który ma być używany podczas komunikacji z menedżerem kolejek, a zestaw CipherSuite musi być zgodny z atrybutem CipherSpec ustawionym na kanale docelowym. Ponadto nazwa CipherSuite musi być obsługiwana przez dostawcę JSSE. Jednak pakiety CipherSuites różnią się od specyfikacji CipherSpecs i mają różne nazwy. Produkt “TLS CipherSpecs i CipherSuites w podręczniku IBM MQ classes for Java” na stronie 406 zawiera tabelę odwzorowania specyfikacji CipherSpecs obsługiwaną przez produkt IBM MQ na równoważną wartość parametru CipherSuites , tak jak jest to znane w przypadku rozszerzenia JSSE.

Aby włączyć obsługę protokołu TLS, należy określić pakiet CipherSuite przy użyciu statycznej zmiennej składowej sslCipherSuite produktu MQEnvironment. Poniższy przykład przyłącza się do kanału SVRCONN o nazwie SECURE.SVRCONN.CHANNEL, który został skonfigurowany tak, aby wymagał protokołu TLS z atrybutem CipherSpec o wartości 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");
```

Mimo że w kanale znajduje się specyfikacja CipherSpec o wartości TLS_RSA_WITH_AES_128_CBC_SHA256, aplikacja Java musi określić pakiet CipherSuite o wartości SSL_RSA_WITH_AES_128_CBC_SHA256. Listę odwzorowań między CipherSpecs i CipherSuitesmożna znaleźć w sekcji “TLS CipherSpecs i CipherSuites w podręczniku IBM MQ classes for Java” na stronie 406 .

Aplikacja może również określić pakiet CipherSuite , ustawiając właściwość środowiska CMQC.SSL_CIPHER_SUITE_PROPERTY.

Alternatywnie można użyć tabeli definicji kanału klienta (CCDT). Więcej informacji na ten temat zawiera sekcja “Korzystanie z tabeli definicji kanału klienta przy użyciu produktu IBM MQ classes for Java” na stronie 372.

If you require a client connection to use a CipherSuite that is supported by the IBM Java JSSE FIPS provider (IBMJSSEFIPS), an application can set the sslFipsRequired field in the MQEnvironment class to `true`. Alternatywnie aplikacja może ustawić właściwość środowiska CMQC.SSL_FIPS_REQUIRED_PROPERTY. Wartością domyślną jest `false`, co oznacza, że połączenie klienckie może używać dowolnego zestawu CipherSuite obsługiwanego przez produkt IBM MQ.

Jeśli aplikacja używa więcej niż jednego połączenia klienckiego, wartość pola sslFipswymaganego, która jest używana, gdy aplikacja tworzy pierwsze połączenie klienckie, określa wartość używaną podczas tworzenia kolejnego połączenia klienckiego przez aplikację. Z tego powodu, gdy aplikacja utworzy kolejne połączenie klienckie, wartość pola sslFipsRequired jest ignorowana. Jeśli w polu sslFipswymagane jest użycie innej wartości, należy zrestartować aplikację.

Aby pomyślnie nawiązać połączenie przy użyciu protokołu TLS, magazyn zaufanych certyfikatów JSSE musi być skonfigurowany z certyfikatami głównego ośrodka certyfikacji, z których certyfikat prezentowany przez menedżer kolejek może zostać uwierzytelniony. Podobnie, jeśli parametr SSLClientAuth w kanale SVRCONN został ustawiony na 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 produktu UNIX, Linux, and Windows

Korzystanie z nazwy wyróżniającej menedżera kolejek w produkcie 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 używać tej nazwy wyróżniającej, aby zapewnić komunikację z poprawnym menedżerem kolejek.

Wzorzec nazwy wyróżniającej jest określany przy użyciu zmiennej nazwy sslPeernazwy środowiska MQEnvironment. Na przykład:

```
MQEnvironment.sslPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSPHERE";
```

Umożliwia nawiązanie połączenia tylko wtedy, gdy menedżer kolejek wyświetli certyfikat o nazwie Common Name rozpoczynający się od QMGR., i co najmniej dwie nazwy jednostek organizacyjnych, z których pierwsza musi być IBM , a druga WebSphere.

Jeśli ustawiona jest nazwa sslPeerNazwa, połączenia powiodą się tylko wtedy, gdy zostanie ustawiona na poprawny wzorzec, a menedżer kolejek wyświetli pasujący 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 produkcie IBM MQ classes for Java
Należy określić listy odwołań certyfikatów, które mają być używane przez klasę java.security.cert.CertStore . Następnie produkt IBM MQ classes for Java sprawdza certyfikaty dla określonej listy CRL.

Lista odwołań certyfikatów (CRL) jest zestawem certyfikatów, które zostały odwołane przez wystawiający ośrodek certyfikacji lub przez organizację lokalną. Listy CRL są zwykle udostępniane na serwerach LDAP. W przypadku systemu Java 2 v1.4serwer listy CRL może zostać określony w czasie połączenia, a certyfikat przedstawiony przez menedżer kolejek jest sprawdzany przed listą CRL przed zezwolniem na nawiązanie połączenia. Więcej informacji na temat list odwołań certyfikatów i IBM MQzawiera sekcja Praca z listami odwołań certyfikatów i listami odwołań uprawnień oraz Uzyskiwanie dostępu do list CRL i ARL za pomocą produktów IBM MQ classes for Java i IBM MQ classes for JMS.

Uwaga: Aby pomyślnie użyć narzędzia CertStore z listą CRL udostępnianą na serwerze LDAP, należy upewnić się, że pakiet Java Software Development Kit (SDK) jest kompatybilny z 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 zamiast niego RFC 2256.

Listy CRL, które mają być używane, są określone za pomocą klasy java.security.cert.CertStore . Szczegółowe informacje na temat uzyskiwania instancji programu CertStoremożna znaleźć w dokumentacji tej klasy. Aby utworzyć obiekt CertStore na podstawie serwera LDAP, należy najpierw utworzyć instancję LDAPCertStore, która zostanie zainicjowana z użyciem ustawień serwera i portu. Na przykład:

```
import java.security.cert.*;
CertStoreParameters csp = new LDAPCertStoreParameters("crl_server", 389);
```

Po utworzeniu instancji parametrów CertStorenależy użyć konstruktora statycznego w składnicy CertStore w celu utworzenia składnicy CertStore typu LDAP:

```
CertStore cs = CertStore.getInstance("LDAP", csp);
```

Obsługiwane są także inne typy CertStore (na przykład Kolekcja). Często istnieje kilka serwerów CRL ustawionych z identycznymi informacjami listy CRL, aby zapewnić nadmiarowość. Jeśli dla każdego z tych serwerów CRL istnieje obiekt CertStore , należy umieścić 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);
```

Ta kolekcja może zostać ustawiona w zmiennej statycznej MQEnvironment sslCertSklepy przed nawiązywaniem połączenia w celu włączenia sprawdzania CRL:

```
MQEnvironment.sslCertStores = crls;
```

Poprawność certyfikatu prezentowanego przez menedżera kolejek przy ustawionym połączeniu jest sprawdzana w następujący sposób:

1. Pierwszy obiekt CertStore w kolekcji identyfikowanej przez sslCertStores służy do identyfikacji serwera CRL.
2. Podjęto próbę skontaktowania się z serwerem CRL.
3. Jeśli próba zakończy się pomyślnie, serwer jest wyszukiwany pod kątem zgodności z certyfikatem.
   a. Jeśli certyfikat zostanie unieważniony, proces wyszukiwania zakończy się, a żądanie nawiązania połączenia nie powiedzie się i zostanie odebrany kod przyczyny MQRC_SSL_CERTIFICATE_ODWOŁANE.
   b. Jeśli certyfikat nie zostanie znaleziony, proces wyszukiwania zostanie nadany, a połączenie może być kontynuowane.
4. Jeśli próba skontaktowania się z serwerem nie powiedzie się, następny obiekt CertStore jest używany do identyfikacji serwera CRL, a proces jest powtarzany z kroku 2.

   Jeśli była to ostatnia CertStore w kolekcji lub jeśli kolekcja nie zawiera obiektów CertStore , proces wyszukiwania nie powiódł się, a żądanie nawiązania połączenia nie powiodło się. Kod przyczyny: MQRC_SSL_CERT_STORE_ERROR.

Obiekt Collection określa kolejność, w jakiej używane są CertStores .

Kolekcja CertStores może być również ustawiona za pomocą CMQC.SSL_CERT_STORE_PROPERTY. Dla wygody ta właściwość umożliwia również określenie pojedynczego obiektu CertStore , który nie jest elementem kolekcji.

Jeśli parametr sslCertStores jest ustawiony na wartość null, sprawdzanie listy CRL nie jest wykonywane. Ta właściwość jest ignorowana, jeśli zestaw sslCipherSuite nie jest ustawiony.

Renegocjowanie klucza tajnego w produkcie IBM MQ classes for Java
Aplikacja kliencka IBM MQ classes for Java może sterować tym, kiedy klucz tajny używany do szyfrowania połączenia klienckiego jest renegocjowany, jeśli chodzi o łączną liczbę wysłanych i odebranych bajtów.

Aplikacja może to zrobić w jeden z następujących sposobów: jeśli aplikacja używa więcej niż jednego z tych sposobów, stosowane są zwykłe reguły dotyczące kolejności wykonywania zadań.

- W tym celu należy ustawić pole sslResetCount w klasie MQEnvironment.
- Ustawiając właściwość środowiska MQC.SSL_RESET_COUNT_PROPERTY w obiekcie Hashtable. Następnie aplikacja przypisuje tabelę mieszającą do pola `properties` w klasie MQEnvironment lub przekazuje tabelę mieszającą do obiektu MQQueueManager na jego konstruktorze.

Wartość pola licznika sslResetlub właściwości środowiska MQC.SSL_RESET_COUNT_PROPERTY reprezentuje łączną liczbę bajtów wysłanych i odebranych przez kod klienta IBM MQ classes for Java przed renegocjacją klucza tajnego. Liczba wysłanych bajtów jest liczbą przed zaszyfrowaniem, a liczba odebranych bajtów jest liczbą po deszyfrowaniu. Liczba bajtów obejmuje również informacje sterujące wysłane i odebrane przez klienta IBM MQ classes for Java .

Jeśli wartość licznika resetowania wynosi zero, co jest wartością domyślną, klucz tajny nigdy nie będzie ponownie negocjowany. Licznik resetowania jest ignorowany, jeśli nie zostanie podany parametr CipherSuite .

Dostarczanie dostosowanej fabryki SSLSocketFactory w produkcie IBM MQ classes for Java
Jeśli używana jest niestandardowa fabryka gniazd JSSE, należy ustawić właściwość MQEnvironment.sslSocketFactory na dostosowany obiekt fabryczny. Szczegóły różnią się między różnymi implementacjami JSSE.

Różne implementacje JSSE mogą udostępniać różne funkcje. Na przykład wyspecjalizowana implementacja JSSE może zezwalać na konfigurację konkretnego modelu sprzętu szyfrującego. Ponadto niektóre dostawcy JSSE umożliwiają dostosowywanie plików kluczy i magazynów zaufanych certyfikatów według programu lub umożliwiają zmianę opcji wyboru certyfikatu tożsamości z magazynu kluczy. W środowisku JSSE wszystkie te dostosowania są streszczane w klasie fabryki, javax.net.ssl.SSLSocketFactory.

Szczegółowe informacje na temat tworzenia dostosowanej implementacji SSLSocketFactory można znaleźć w dokumentacji JSSE. Szczegóły różnią się od dostawcy do dostawcy, ale typowa sekwencja kroków może być następująca:

1. Tworzenie obiektu SSLContext przy użyciu metody statycznej przy użyciu kontekstu SSLContext.
2. Zainicjuj ten kontekst SSLContext przy użyciu odpowiednich implementacji KeyManager i TrustManager (utworzonych na podstawie ich własnych klas fabrycznych)
3. Utwórz obiekt SSLSocketFactory z kontekstu SSLContext.

Jeśli istnieje obiekt SSLSocketFactory , ustaw właściwość MQEnvironment.sslSocketFactory na dostosowany obiekt fabryczny. Na przykład:

```
javax.net.ssl.SSLSocketFactory sf = sslContext.getSocketFactory();
MQEnvironment.sslSocketFactory = sf;
```

IBM MQ classes for Java użyj tej komendy SSLSocketFactory , aby połączyć się z menedżerem kolejek produktu IBM MQ . Tę właściwość można również ustawić przy użyciu właściwości CMQC.SSL_SOCKET_FACTORY_PROPERTY. Jeśli właściwość sslSocketFactory jest ustawiona na wartość NULL, używana jest domyślna fabryka SSLSocketFactory maszyny JVM. Ta właściwość jest ignorowana, jeśli zestaw sslCipherSuite nie jest ustawiony.

Jeśli używana jest niestandardowa właściwość SSLSocketFactories, należy wziąć pod uwagę wpływ współużytkowania połączeń TCP/IP. Jeśli możliwe jest współużytkowanie połączeń, nie jest wymagane podanie nowego gniazda dla podanej fabryki SSLSocketFactory , nawet jeśli w kontekście kolejnego żądania połączenia wygenerowane gniazdo będzie inne niż w jakiś sposób. Na przykład, jeśli w kolejnym połączeniu ma zostać przedstawiony inny certyfikat klienta, współużytkowanie połączeń nie może być dozwolone.

Wprowadzanie zmian w magazynie kluczy JSSE lub magazynie zaufanych certyfikatów w produkcie IBM MQ classes for Java
Jeśli zmienisz magazyn kluczy lub magazyn zaufanych certyfikatów JSSE, musisz wykonać określone działania, aby zmiany zostały uwzględnione.

Jeśli zawartość magazynu kluczy lub magazynu zaufanych certyfikatów JSSE zostanie zmieniona lub zostanie zmieniona lokalizacja pliku kluczy lub pliku zaufanych certyfikatów, wówczas aplikacje produktu IBM MQ classes for Java , które są uruchomione w danym momencie, nie pobierają automatycznie zmian. Aby zmiany zostały uwzględnione, muszą zostać wykonane następujące działania:

- Aplikacje muszą zamknąć wszystkie połączenia i zniszczyć wszelkie nieużywane połączenia w pulach połączeń.
- Jeśli dostawca JSSE buforuje informacje z magazynu kluczy i magazynu zaufanych certyfikatów, te informacje muszą zostać odświeżone.

Po wykonaniu tych czynności aplikacje mogą następnie ponownie utworzyć połączenia.

W zależności od sposobu zaprojektowania aplikacji oraz funkcji udostępnianej przez dostawcę JSSE możliwe jest wykonanie tych działań bez zatrzymywania i restartowania aplikacji. Jednak zatrzymywanie i restartowanie aplikacji może być najprostszym rozwiązaniem.

Obsługa błędów podczas używania protokołu TLS z produktem IBM MQ classes for Java
IBM MQ classes for Java podczas nawiązywania połączenia z menedżerem kolejek przy użyciu protokołu TLS może zostać wydana pewna liczba kodów przyczyny.

Wyjaśnienia te przedstawiono na poniższej liście:

MQRC_SSL_NOT_ALLOWED
Ustawiono właściwość pakietu sslCipherSuite, ale użyto połączenia powiązań. Tylko klient łączy obsługę protokołu TLS.

MQRC_JSSE_ERROR
Dostawca JSSE zgłosił błąd, który nie może być obsługiwany przez produkt IBM MQ. Może to być spowodowane problemem konfiguracji z rozszerzeniem JSSE lub nie powiodło się sprawdzenie

poprawności certyfikatu przedstawionego przez menedżer kolejek. Wyjątek utworzony przez JSSE można pobrać za pomocą metody getCause() w wyjątku MQException.

MQRC_SSL_INITIALIZATION_ERROR,
Wywołano komendę 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 sslPeerName nie jest zgodny z nazwą wyróżniającą podaną przez menedżer kolejek.

MQRC_SSL_PEER_NAME_ERROR-BŁĄD
Wzorzec nazwy wyróżniającej określony we właściwości sslPeerName nie był poprawny.

MQRC_UNSUPPORTED_CIPHER_SUITE
Pakiet CipherSuite nazwany w pakiecie sslCipherSuite nie został rozpoznany przez dostawcę JSSE. Pełna lista pakietów CipherSuites obsługiwana przez dostawcę JSSE może zostać uzyskana przez program przy użyciu metody SSLSocketFactory.getSupportedCipherSuites(). Listę pakietów CipherSuites , które mogą być używane do komunikacji z produktem IBM MQ , można znaleźć w składniku “TLS CipherSpecs i CipherSuites w podręczniku IBM MQ classes for Java” na stronie 406.

MQRC_SSL_CERTIFICATE_ODWOŁANE
Certyfikat prezentowany przez menedżer kolejek został znaleziony w CRL określonej za pomocą właściwości sslCertStores. Zaktualizuj menedżer kolejek, aby używać zaufanych certyfikatów.

MQRC_SSL_CERT_STORE_ERROR
Żaden z dostarczonych obiektów CertStores nie może być przeszukiwany pod kątem certyfikatu prezentowanego przez menedżera kolejek. Metoda MQException.getCause() zwraca błąd, który wystąpił podczas wyszukiwania pierwszej próby wykonania komendy CertStore . Jeśli wyjątek przyczynowy to NoSuchElementException, ClassCastException lub NullPointerException, należy sprawdzić, czy kolekcja określona we właściwości sslCertStores zawiera co najmniej jeden poprawny obiekt CertStore .

TLS CipherSpecs i CipherSuites w podręczniku IBM MQ classes for Java
Zdolność aplikacji IBM MQ classes for Java do nawiązywania połączeń z menedżerem kolejek zależy od specyfikacji CipherSpec określonej na końcu serwera kanału MQI i CipherSuite określonej na końcu klienta.

W poniższej tabeli znajduje się lista specyfikacji CipherSpecs obsługiwanych przez produkt IBM MQ oraz ich odpowiedniki CipherSuites.

Należy przejrzeć temat Nieaktualne CipherSpecs , aby sprawdzić, czy którykolwiek z CipherSpecs, wymieniony w poniższej tabeli, jest nieaktualny w produkcie IBM MQ , a jeśli tak, to przy czym aktualizacja specyfikacji CipherSpec była nieaktualna.

Ważne: wymienione w liście CipherSuites są obsługiwane przez środowisko wykonawcze IBM Java Runtime Environment (JRE) dostarczane z produktem IBM MQ. Wymienione pakiety CipherSuites obejmują te, które są obsługiwane przez środowisko JRE firmy Oracle Java . 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 436.

Tabela wskazuje również protokół używany do komunikacji, niezależnie od tego, czy pakiet CipherSuite jest zgodny ze standardem FIPS 140-2.

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 zgodność ze standardem FIPS 140-2 została skonfigurowana dla aplikacji (patrz następujące uwagi w konfiguracji), można skonfigurować tylko te CipherSuites , które są oznaczone jako zgodne ze standardem FIPS 140-2. Próba użycia innych pakietów CipherSuites powoduje błąd.

Uwaga: Każde środowisko JRE może mieć wiele dostawców zabezpieczeń kryptograficznych, z których każdy może przyczyniać się do implementacji tego samego pakietu CipherSuite. Jednak nie wszyscy dostawcy zabezpieczeń są certyfikowani zgodnie ze standardem FIPS 140-2. Jeśli zgodność ze standardem FIPS 140-2 nie jest wymuszana dla aplikacji, możliwe jest użycie niecertyfikowanej

implementacji pakietu CipherSuite . Niecertyfikowane implementacje mogą nie działać zgodnie ze standardem FIPS 140-2, nawet jeśli teoretycznie CipherSuite spełnia minimalny poziom zabezpieczeń wymagany przez standard. Więcej informacji na temat konfigurowania wymuszania FIPS 140-2 w aplikacjach IBM MQ Java zawierają następujące uwagi.

Więcej informacji na temat zgodności ze standardem FIPS 140-2 i Suite-B dla specyfikacji CipherSpecs i CipherSuiteszawiera sekcja Określanie specyfikacji CipherSpecs. Konieczne może być również zapoznanie się z informacjami, które dotyczą Stanów Zjednoczonych Federal Information Processing Standards(Federal Information Processing Standards).

Aby korzystać z pełnego zestawu pakietów CipherSuites i obsługiwać zgodność ze standardem FIPS 140-2 i/lub Suite-B, wymagane jest odpowiednie środowisko JRE. IBM Produkt Java 7 Service Refresh 4 z pakietem poprawek Fix Pack 2 lub wyższy poziom środowiska IBM JRE zapewnia odpowiednią obsługę protokołu TLS 1.2 CipherSuites wymienionego w sekcji Tabela 58 na stronie 408.

V 9.1.5 Aby możliwe było użycie szyfrowania TLS v1.3 , środowisko JRE, na którym działa aplikacja, musi obsługiwać protokół TLS v1.3.

Uwaga: Aby można było używać niektórych pakietów CipherSuites, w środowisku JRE należy skonfigurować pliki strategii 'nieograniczone'. Więcej szczegółowych informacji na temat konfigurowania plików strategii w pakiecie SDK lub środowisku JRE zawiera temat *Pliki strategii pakietu SDK produktu IBM* w publikacji *Skorowidz zabezpieczeń dla pakietu IBM SDK, Java Technology Edition* dla używanej wersji.

Tabela 58. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**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 58. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**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 58. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**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 58. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**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 58. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**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 58. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**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

Tabela 58. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**Odpowiednik CipherSuite (Oracle JRE)**	**Protokół**	**Zgodny ze standardem FIPS 140-2**
ECDHE_ECDSA_RC4_128_SHA256	SSL_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	TLS 1.2	no

Tabela 58. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**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 58. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**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 58. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**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 58. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**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 58. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**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 58. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**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

Tabela 58. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**Odpowiednik CipherSuite (Oracle JRE)**	**Protokół**	**Zgodny ze standardem FIPS 140-2**
ECDHE_RSA_RC4_128_SHA256	SSL_ECDHE_RSA_WITH_RC4_128_SHA	TLS_ECDHE_RSA_WITH_RC4_128_SHA	TLS 1.2	no

Tabela 58. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	Odpowiednik CipherSuite (IBM JRE)	Odpowiednik CipherSuite (Oracle JRE)	Protokół	Zgodny ze standardem FIPS 140-2
TLS_RSA_WITH_3DES_EDE_CBC_SHA "1" na stronie 435	SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA	TLS 1.0	yes

Tabela 58. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**Odpowiednik CipherSuite (Oracle JRE)**	**Protokół**	**Zgodny ze standardem FIPS 140-2**
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA	TLS_RSA_WITH_AES_128_CBC_SHA	TLS 1.0	yes

Tabela 58. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**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	yes

Tabela 58. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**Odpowiednik CipherSuite (Oracle JRE)**	**Protokół**	**Zgodny ze standardem FIPS 140-2**
TLS_RSA_WITH_AES_128_GCM_SHA256	SSL_RSA_WITH_AES_128_GCM_SHA256	TLS_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2	yes

Tabela 58. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**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	yes

Tabela 58. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**Odpowiednik CipherSuite (Oracle JRE)**	**Protokół**	**Zgodny ze standardem FIPS 140-2**
TLS_RSA_WITH_AES_256_CBC_SHA256	SSL_RSA_WITH_AES_256_CBC_SHA256	TLS_RSA_WITH_AES_256_CBC_SHA256	TLS 1.2	yes

Tabela 58. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**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	yes

Tabela 58. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**Odpowiednik CipherSuite (Oracle JRE)**	**Protokół**	**Zgodny ze standardem FIPS 140-2**
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	TLS 1.0	no

Tabela 58. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**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

Tabela 58. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**Odpowiednik CipherSuite (Oracle JRE)**	**Protokół**	**Zgodny ze standardem FIPS 140-2**
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 58. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	Odpowiednik CipherSuite (IBM JRE)	Odpowiednik CipherSuite (Oracle JRE)	Protokół	Zgodny ze standardem FIPS 140-2
V 9.1.5 TLS_AES_128_GCM_SHA256 “2” na stronie 436	TLS_AES_128_GCM_SHA256	TLS_AES_128_GCM_SHA256	TLS 1.3	no
V 9.1.5 TLS_AES_256_GCM_SHA384 “2” na stronie 436	TLS_AES_256_GCM_SHA384	TLS_AES_256_GCM_SHA384	TLS 1.3	no

Tabela 58. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**Odpowiednik CipherSuite (Oracle JRE)**	**Protokół**	**Zgodny ze standardem FIPS 140-2**
V 9.1.5 TLS_CHACHA20_POLY1305_SHA256 "2" na stronie 436	TLS_CHACHA20_POLY1305_SHA256	TLS_CHACHA20_POLY1305_SHA256	TLS 1.3	no

Tabela 58. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	**Odpowiednik CipherSuite (IBM JRE)**	**Odpowiednik CipherSuite (Oracle JRE)**	**Protokół**	**Zgodny ze standardem FIPS 140-2**
V 9.1.5 TLS_AES_128_CCM_SHA256 “2” na stronie 436	TLS_AES_128_CCM_SHA256	TLS_AES_128_CCM_SHA256	TLS 1.3	no
V 9.1.5 TLS_AES_128_CCM_8_SHA256 “2” na stronie 436	TLS_AES_128_CCM_8_SHA256	TLS_AES_128_CCM_8_SHA256	TLS 1.3	no

Tabela 58. CipherSpecs obsługiwane przez produkt IBM MQ i ich odpowiedniki CipherSuites (kontynuacja)

CipherSpec	Odpowiednik CipherSuite (IBM JRE)	Odpowiednik CipherSuite (Oracle JRE)	Protokół	Zgodny ze standardem FIPS 140-2
V 9.1.5 Dowolna "2" na stronie 436	*ANY	*ANY	Wielokrotny	no
V 9.1.5 ANY_TLS13 "2" na stronie 436	*TLS13	*TLS13	TLS V13	no
V 9.1.5 ANY_TLS12_OR_HIGHER "2" na stronie 436	*TLS12ORHIGHER	*TLS12ORHIGHER	TLS V1.2 i nowsze	no
V 9.1.5 ANY_TLS13_OR_HIGHER "2" na stronie 436	*TLS13ORHIGHER	*TLS13ORHIGHER	TLS V1.3 i nowsze	no

Uwagi:

1. Klasa CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA jest nieaktualna. Jednak może być ona nadal używana do przesyłania do 32 GB danych, zanim połączenie zostanie zakończone z błędem AMQ9288.

Aby uniknąć tego błędu, należy unikać używania potrójnego algorytmu szyfrowania DES lub włączyć resetowanie klucza tajnego podczas korzystania z tej specyfikacji CipherSpec.

2. V 9.1.5 Aby możliwe było użycie szyfrowania TLS v1.3 , środowisko JRE, na którym działa aplikacja, musi obsługiwać protokół TLS v1.3 .

Konfigurowanie pakietów Ciphersuites i zgodności ze standardem FIPS w aplikacji IBM MQ classes for Java

- Aplikacja, która używa produktu IBM MQ classes for Java , może użyć jednej z dwóch metod w celu ustawienia pakietu CipherSuite dla połączenia:
  - Ustaw wartość w polu sslCipherSuite w klasie MQEnvironment na nazwę CipherSuite .
  - Ustaw właściwość CMQC.SSL_CIPHER_SUITE_PROPERTY w tabeli mieszającej właściwości przekazanej do konstruktora MQQueueManager na nazwę CipherSuite .
- Aplikacja, która używa produktu IBM MQ classes for Java , może użyć jednej z dwóch metod w celu wymuszenia zgodności ze standardem FIPS 140-2:
  - W klasie MQEnvironment ustaw wymagane pole sslFipsna wartość true.
  - Ustaw właściwość CMQC.SSL_FIPS_REQUIRED_PROPERTYin w tabeli mieszającej właściwości przekazanej do konstruktora MQQueueManager na wartość true.

Konfigurowanie aplikacji do korzystania z odwzorowań IBM Java lub Oracle Java CipherSuite

Istnieje możliwość skonfigurowania, czy aplikacja używa domyślnych odwzorowań IBM Java CipherSuite do IBM MQ CipherSpec , czy Oracle CipherSuite do IBM MQ CipherSpec odwzorowań. Oznacza to, że można używać protokołu TLS CipherSuites , niezależnie od tego, czy aplikacja korzysta ze środowiska IBM JRE, czy środowiska Oracle JRE. Właściwość systemowa Java `com.ibm.mq.cfg.useIBMCipherMappings` określa, które odwzorowania są używane. Właściwość może mieć jedną z następujących wartości:

true
Użyj opcji IBM Java CipherSuite do IBM MQ CipherSpec odwzorowań.

Ta wartość jest wartością domyślną.

Fałsz
Use the Oracle CipherSuite to IBM MQ CipherSpec mappings.

Więcej informacji na temat korzystania z programów IBM MQ Java i TLS Ciphers zawiera blog MQdev: MQ Java, TLS Ciphers, Non-IBM JREs & APARs IT06775, IV66840, IT09423, IT10837.

Ograniczenia współdziałania

Niektóre pakiety CipherSuites mogą być kompatybilne z więcej niż jednym IBM MQ CipherSpec, w zależności od tego, jaki protokół jest używany. Jednak obsługiwane jest tylko połączenie CipherSuite/CipherSpec , które używa wersji TLS określonej w tabeli 1. Próba użycia nieobsługiwanych kombinacji CipherSuites i CipherSpecs nie powiedzie się i zostanie zgłoszony odpowiedni wyjątek. Instalacje korzystające z dowolnego z tych kombinacji CipherSuite/CipherSpec powinny przenieść się do obsługiwanej kombinacji.

W poniższej tabeli przedstawiono pakiety CipherSuites , do których ma zastosowanie to ograniczenie.

Tabela 59. CipherSuites i ich obsługiwane i nieobsługiwane CipherSpecs		
CipherSuite	**Obsługiwany protokół TLS CipherSpec**	**Nieobsługiwany protokół SSL CipherSpec**
SSL_RSA_WITH_3DES_EDE_CBC_SHA	TLS_RSA_WITH_3DES_EDE_CBC_SHA "1" na stronie 437	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. Ta specyfikacja CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA jest nieaktualna. Jednak może być ona nadal używana do przesyłania do 32 GB danych, zanim połączenie zostanie zakończone z błędem AMQ9288. Aby uniknąć tego błędu, należy unikać używania potrójnego algorytmu szyfrowania DES lub włączyć resetowanie klucza tajnego podczas korzystania z tej specyfikacji CipherSpec.

Uruchamianie aplikacji produktu IBM MQ classes for Java

W przypadku napisania aplikacji (klasy zawierającej metodę main ()), przy użyciu trybu klienta lub powiązań, należy uruchomić program przy użyciu interpretera Java .

Użyj komendy:

```
java -Djava.library.path= library_path MyClass
```

gdzie *ścieżka_biblioteki* jest ścieżką do bibliotek produktu IBM MQ classes for Java . Więcej informacji na ten temat zawiera sekcja "IBM MQ classes for Java biblioteki" na stronie 353.

Zadania pokrewne

Śledzenie aplikacji IBM MQ classes for Java

Śledzenie adaptera zasobów produktu IBM MQ

IBM MQ classes for Java zachowanie zależne od środowiska

Produkt IBM MQ classes for Java umożliwia tworzenie aplikacji, które mogą być uruchamiane dla różnych wersji produktu IBM MQ. Ta kolekcja tematów opisuje zachowanie klas Java zależnych od tych różnych wersji.

Produkt IBM MQ classes for Java udostępnia podstawowe klasy, które zapewniają spójną funkcję i zachowanie we wszystkich środowiskach. Funkcje poza tym rdzeniem zależą od możliwości menedżera kolejek, z którym połączona jest aplikacja.

Z wyjątkiem sytuacji, w których w tym miejscu wspomniano, zachowanie jest opisane w odwołaniu do aplikacji MQI odpowiednich dla menedżera kolejek.

Klasy podstawowe w produkcie IBM MQ classes for Java

Produkt IBM MQ classes for Java zawiera podstawowy zestaw klas, które mogą być używane we wszystkich środowiskach.

Następujący zestaw klas jest uważany za klasy podstawowe i może być używany we wszystkich środowiskach, w których znajdują się tylko niewielkie różnice wymienione w sekcji "Ograniczenia i warianty klas podstawowych produktu IBM MQ classes for Java" na stronie 439.

- Środowisko MQEnvironment
- Wyjątek MQException
- Opcje MQGetMessage

  Wykluczanie:

  - MatchOptions
  - GroupStatus
  - SegmentStatus
  - Segmentacja
- MQManagedObject

  Wykluczanie:
  - inquire ()
  - set ()
- Komunikat MQMessage

  Wykluczanie:
  - groupId
  - messageFlags
  - messageSequenceLiczba
  - Przesunięcie
  - originalLength
- MQPoolServices
- Zdarzenie MQPoolServices
- MQPoolServicesEventListener
- MQPoolToken
- Opcje MQPutMessage

  Wykluczanie:
  - Liczba knownDest
  - Liczba unknownDest
  - Liczba invalidDest
  - recordFields
- Proces MQProcess
- MQQUEUE
- MQQueueManager

  Wykluczanie:
  - początek ()
  - Lista accessDistribution()
- Menedżer MQSimpleConnection
- Temat MQTopic
- MQC

Uwaga:

1. Niektóre stałe nie są zawarte w rdzeniu (szczegółowe informacje na ten temat znajdują się w sekcji “Ograniczenia i warianty klas podstawowych produktu IBM MQ classes for Java” na stronie 439). Nie należy ich używać w programach przenośnych.
2. Niektóre platformy nie obsługują wszystkich trybów połączenia. Na tych platformach można używać tylko klas i opcji podstawowych, które odnoszą się do obsługiwanych trybów.

Ograniczenia i warianty klas podstawowych produktu IBM MQ classes for Java

Klasy podstawowe zwykle zachowują się w sposób spójny we wszystkich środowiskach, nawet jeśli równoważne wywołania MQI zwykle mają różnice w środowisku. The behavior is as if a Windows, UNIX or Linux IBM MQ queue manager is used, except for the following minor restrictions and variations.

*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 obiektu MQQueue.get():

MQGMO_SYNCPOINT_IF_PERSISTENT
MQGMO_MARK_SKIP_BACKOUT
MQGMO_BROWSE_MSG_UNDER_CURSOR
Blokada MQGMO_LOCK
MQGMO_UNLOCK
MQGMO_LOGICAL_ORDER
MQGMO_COMPLETE_MESSAGE
MQGMO_ALL_MSGS_AVAILABLE
MQGMO_ALL_SEGMENTS_AVAILABLE
MQGMO_UNMARKED_BROWSE_MSG
MQGMO_MARK_BROWSE_HANDLE
MQGMO_MARK_BROWSE_CO_OP
MQGMO_UNMARK_BROWSE_HANDLE
MQGMO_UNMARK_BROWSE_CO_OP

Dodatkowo, parametr MQGMO_SET_SIGNAL nie jest obsługiwany w przypadku użycia 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 produktu 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 obiektu MQQueue.put() lub MQQueueManager.put ():

MQPMO_LOGICAL_ORDER
MQPMO_NEW_CORREL_ID
MQPMO_NEW_MESSAGE_ID
MQPMO_RESOLVE_LOCAL_Q

*Ograniczenia i zmiany wartości MQCNO_ * w produkcie IBM MQ classes for Java*
Niektóre wartości `MQCNO_*` nie są obsługiwane.

- Automatyczne ponowne łączenie klienta nie jest obsługiwane przez produkt IBM MQ classes for Java. Niezależnie od ustawionej wartości `MQCNO_RECONNECT_*` połączenie zachowuje się tak, jak w przypadku ustawienia `MQCNO_RECONNECT_DISABLED`.
- Produkt `MQCNO_FASTPATH` jest ignorowany w menedżerach kolejek, które nie obsługują produktu `MQCNO_FASTPATH`. Jest ona również ignorowana 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_COA_WITH_FULL_DATA

MQRO_COD_WITH_FULL_DATA
MQRO_DISCARD_MSG
MQRO_PASS_DISCARD_AND_WAŻNOŚCI

Więcej informacji na ten temat zawiera sekcja Raport.

Różne różnice między produktem IBM MQ classes for Java na serwerze z/OS i innymi platformami
Produkt IBM MQ for z/OS zachowuje się inaczej niż produkt IBM MQ na innych platformach w niektórych obszarach.

BackoutCount
Menedżer kolejek produktu z/OS zwraca wartość maksymalną BackoutCount równą 255, nawet jeśli komunikat został wycofany z pamięci 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 MQCC_FAILED i MQRC_ENVIRONMENT_ERROR.

Konstruktor MQQueueManager może również zakończyć się niepowodzeniem z błędem MQRC_CHAR_CONVERSION_ERROR (jeśli nie powiodło się zainicjowanie konwersji między stronami kodowym IBM-1047 i ISO8859-1) lub MQRC_UCS2_CONVERSION_ERROR (jeśli nie powiodło się zainicjowanie konwersji między stroną kodową menedżera kolejek a Unicode). Jeśli aplikacja nie powiedzie się z jednym z tych kodów przyczyny, upewnij się, że zainstalowano komponent National Language Resources dla środowiska językowego, a także upewnij się, że dostępne są poprawne tabele konwersji.

Tabele konwersji dla kodu Unicode są instalowane jako część opcjonalnej funkcji produktu z/OS C/C + +. Więcej informacji na temat włączania konwersji UCS-2 można znaleźć w podręczniku *z/OS C/C++ Programming Guide*(SC09-4765).

Składniki znajdujące się poza klasami podstawowymi produktu IBM MQ classes for Java

Produkt IBM MQ classes for Java zawiera pewne funkcje, które zostały zaprojektowane w taki sposób, aby używać rozszerzeń interfejsu API, które nie są obsługiwane przez wszystkie menedżery kolejek. Ta kolekcja tematów opisuje sposób zachowania się podczas korzystania z menedżera kolejek, który ich nie obsługuje.

Zmiany w opcji konstruktora MQQueueManager
Niektóre konstruktory produktu MQQueueManager zawierają opcjonalny argument będący liczbą całkowitą. Niektóre wartości tego argumentu nie są akceptowane na wszystkich platformach.

W przypadku, gdy konstruktor MQQueueManager zawiera opcjonalny argument całkowitoliczbowy, jest on odwzorowyany na pole opcji MQCNO interfejsu MQI i jest używany do przełączania się między normalnym i szybkim połączeniem ścieżki. Ta rozszerzona forma konstruktora jest akceptowana we wszystkich środowiskach, jeśli jedynymi używanymi opcjami są: MQCNO_STANDARD_BINDING lub MQCNO_FASTPATH_BINDING. Inne opcje powodują, że konstruktor nie powiedzie się i zostanie zakończony błąd MQRC_OPTIONS_ERROR. Opcja krótkiej ścieżki CMQC.MQCNO_FASTPATH_BINDING jest honorowane tylko z powiązaniami powiązania z menedżerem kolejek, który go obsługuje. W innych środowiskach jest on ignorowany.

Ograniczenia dotyczące metody MQQueueManager.begin ()
Ta metoda może być używana tylko w przypadku menedżera kolejek produktu IBM MQ w systemach UNIX, Linuxlub Windows w trybie powiązań. W przeciwnym razie błąd komendy MQRC_ENVIRONMENT_ERROR nie powiedzie się.

Więcej informacji na temat zawiera sekcja “Koordynacja JTA/JDBC przy użyciu produktu IBM MQ classes for Java” na stronie 397.

Zmiany w polach opcji MQGetMessage
Niektóre menedżery kolejek nie obsługują struktury MQGMO w wersji 2, dlatego konieczne jest ustawienie niektórych pól na ich wartości domyślne.

W przypadku korzystania z menedżera kolejek, który nie obsługuje struktury MQGMO w wersji 2, należy pozostawić następujące pola ustawione na wartości domyślne:

GroupStatus
SegmentStatus
Segmentacja

Ponadto w polu MatchOptions obsługiwane są tylko parametry MQMO_MATCH_MSG_ID i MQMO_MATCH_CORREL_ID. Jeśli w tych polach zostaną umieszczone nieobsługiwane wartości, to kolejna operacja MQDestination.get() nie powiedzie się i zostanie ona zakończona niepowodzeniem z błędem MQRC_GMO_ERROR. Jeśli menedżer kolejek nie obsługuje struktury MQGMO w wersji 2, to pola te nie są aktualizowane po pomyślnym zakończeniu operacji MQDestination.get().

Ograniczenia w listach dystrybucyjnych w produkcie IBM MQ classes for Java
Nie wszystkie menedżery kolejek umożliwiają otwarcie listy MQDistributionList.

Do tworzenia list dystrybucyjnych używane są następujące klasy:

MQDistributionList
Element MQDistributionList
MQMessageTracker

Istnieje możliwość utworzenia i zapełnienia elementów MQDistributionLists i MQDistributionListw dowolnym środowisku, ale nie wszystkie menedżery kolejek umożliwiają otwarcie listy MQDistributionList. W szczególności menedżery kolejek produktu z/OS nie obsługują list dystrybucyjnych. Próba otwarcia listy MQDistributionList podczas korzystania z takiego menedżera kolejek powoduje błąd MQRC_OD_ERROR.

Zmiany w polach opcji MQPutMessage
Jeśli menedżer kolejek nie obsługuje list dystrybucyjnych, niektóre pola MQPMO są traktowane inaczej.

Cztery pola w strukturze MQPMO są renderowane jako następujące zmienne składowe w klasie opcji MQPutMessage:

Liczba knownDest
Liczba unknownDest
Liczba invalidDest
recordFields

Te pola są przede wszystkim przeznaczone do użycia z listami dystrybucyjną. Jednak menedżer kolejek, który obsługuje listy dystrybucyjne, wypełnia pola DestCount po operacji MQPUT w jednej kolejce. Na przykład, jeśli kolejka jest tłumaczona na kolejkę lokalną, liczba knownDestjest ustawiona na 1, a pozostałe dwa pola licznika są ustawione na 0.

Jeśli menedżer kolejek nie obsługuje list dystrybucyjnych, wartości te są symulowane w następujący sposób:

- Jeśli operacja put () powiedzie się, wartość unknownDestCount jest ustawiona na 1, a pozostałe są ustawione na 0.
- Jeśli wykonanie komendy put () nie powiedzie się, wartość invalidDestLiczba jest ustawiona na 1, a pozostałe są ustawione na 0.

Zmienna recordFields jest używana z listami dystrybucyjną. Wartość może być zapisana w recordFields w dowolnym momencie, niezależnie od środowiska. Jest ona ignorowana, jeśli obiekt opcji MQPutMessagejest używany w kolejnych MQDestination.put() lub MQQueueManager.put (), a nie MQDistributionList.put ().

Ograniczenia w polach MQMD z IBM MQ classes for Java
Niektóre pola MQMD, których dotyczy segmentacja komunikatów, powinny być pozostawiane w wartości domyślnej, gdy używany jest menedżer kolejek, który nie obsługuje segmentacji.

Następujące pola MQMD są w dużej mierze związane z segmentacją komunikatu:

GroupId
Numer_kolejny_komunikatu
Depozycja
MsgFlags
OriginalLength

Jeśli aplikacja ustawia dowolne z tych pól MQMD na wartości inne niż wartości domyślne, a następnie wykonuje komendę put () lub get () dla menedżera kolejek, który nie obsługuje tych pól, to put () lub get () zgłasza wyjątek MQException z MQRC_MD_ERROR. Pomyślne ustawienie put () lub get () z takim menedżerem kolejek zawsze pozostawia pola MQMD ustawione na wartości domyślne. Nie wysyłaj zgrupowanego ani segmentowanego komunikatu do aplikacji Java , która jest uruchamiana dla menedżera kolejek, który nie obsługuje grupowania komunikatów i segmentacji.

Jeśli aplikacja Java próbuje uzyskać () komunikat z menedżera kolejek, który nie obsługuje tych pól, a komunikat fizyczny, który ma zostać pobrany, jest częścią grupy komunikatów segmentowanych (czyli ma wartości inne niż domyślne dla pól MQMD), jest on pobierany bez błędów. Jednak pola MQMD w komunikacie MQMessage nie są aktualizowane. Właściwość formatu MQMessage jest ustawiona na wartość MQFMT_MD_EXTENSION, a prawdziwe dane komunikatu są poprzedzane strukturą MQMDE, która zawiera wartości dla nowych pól.

Ograniczenia dotyczące produktu IBM MQ classes for Java w produkcie CICS Transaction Server

W środowisku serwera CICS Transaction Server for z/OS tylko główny (pierwszy) wątek może wydawać wywołania CICS lub IBM MQ .

Należy zauważyć, że klasy IBM MQ JMS nie są obsługiwane w przypadku aplikacji CICS Java .

Dlatego nie jest możliwe współużytkowanie obiektów MQQueueManager i MQQueue między wątkami w tym środowisku lub utworzenie nowego menedżera MQQueueManager w wątku potomnym.

z/OS Produkt “Różne różnice między produktem IBM MQ classes for Java na serwerze z/OS i innymi platformami” na stronie 440 identyfikuje pewne ograniczenia i odmiany, które mają zastosowanie do IBM MQ classes for Java w przypadku uruchamiania w przypadku menedżera kolejek produktu z/OS . Dodatkowo, w przypadku uruchamiania w ramach produktu CICSmetody sterowania transakcjami w programie MQQueueManager nie są obsługiwane. Zamiast wydawania komendy MQQueueManager.commit () lub MQQueueManager.backout () aplikacje korzystają z 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 uzyskiwanie dostępu do zasobów produktu IBM MQ . Obsługuje komunikację przychodzącą i wychodzącą.

Co zawiera adapter zasobów

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 implementuje interfejsy JCA 1.7 i zawiera IBM MQ classes for JMS. Umożliwia on aplikacjom produktu JMS i komponentami bean sterowanymi 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/subskrybowania.

Adapter zasobów produktu IBM MQ obsługuje dwa typy komunikacji między aplikacją a menedżerem kolejek:

Komunikacja wychodząca
Aplikacja uruchamia połączenie z menedżerem kolejek, a następnie wysyła komunikaty produktu JMS do miejsc docelowych produktu JMS i odbiera komunikaty produktu JMS z miejsc docelowych produktu JMS w sposób synchroniczny.

Komunikacja przychodząca
Komunikat JMS , który dociera do miejsca docelowego produktu 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 korzystanie z interfejsu API produktu IBM MQ classes for Java podczas uzyskiwania dostępu do zasobów menedżera kolejek produktu IBM MQ .

Korzystanie z IBM MQ classes for Java w środowisku Java EE jest obsługiwane z ograniczeniami. Więcej informacji na temat tych ograniczeń można znaleźć w sekcji “Uruchamianie aplikacji produktu IBM MQ classes for Java w produkcie Java EE” na stronie 345.

Która wersja adaptera zasobów jest używana

Wersja produktu Java Platform, Enterprise Edition (Java EE) używanego serwera aplikacji określa wersję adaptera zasobów, która musi być używana:

Java EE 7
Adapter zasobów w wersji IBM MQ 8.0 i nowszych obsługuje produkt JCA v1.7 i udostępnia obsługę produktu JMS 2.0 . Ten adapter zasobów musi być wdrożony na serwerze aplikacji w wersji Java EE 7 i nowszej (patrz “Obsługa instrukcji adaptera zasobów IBM MQ” na stronie 444).

Adapter zasobów produktu IBM MQ 8.0 lub nowszy można zainstalować na dowolnym serwerze aplikacji, który jest certyfikowany jako zgodny ze specyfikacją Java Platform, Enterprise Edition 7 . Za pomocą adaptera zasobów w wersji IBM MQ 8.0 lub nowszej aplikacja może łączyć się z menedżerem kolejek produktu IBM WebSphere MQ 7.0 lub nowszego, korzystając z transportu BINDINGS lub CLIENT albo do menedżera kolejek produktu IBM WebSphere MQ 6.0 wyłącznie przy użyciu transportu CLIENT.

Ważne: Adapter zasobów produktu IBM MQ 8.0 lub nowszy można wdrożyć tylko na serwerze aplikacji, który obsługuje produkt JMS 2.0.

Java EE 5 i Java EE 6
Adapter zasobów produktu IBM WebSphere MQ 7.5 obsługuje produkt Java EE Connector Architecture (JCA) v1.5 i udostępnia obsługę produktu JMS 1.1 . Aby zapewnić pełną integrację z produktem WebSphere Liberty, adapter zasobów produktu IBM WebSphere MQ 7.5 jest aktualizowany do raportu APAR IC92914 z produktu IBM WebSphere MQ 7.5.0 Fix Pack 2. Ten adapter zasobów zachowuje pełną kompatybilność z innymi serwerami aplikacji Java EE 5 i nowszymi (patrz WebSphere MQ resource adapter v7.1 and later statement of support).

Korzystanie z adaptera zasobów z produktem WebSphere Application Server traditional

Z poziomu produktu IBM MQ 9.0adapter zasobów produktu IBM MQ jest wstępnie instalowany w produkcie WebSphere Application Server traditional 9.0 lub nowszym. Z tego powodu nie ma potrzeby instalowania nowego adaptera zasobów.

Uwaga: Adapter zasobów w wersji IBM MQ 9.0 lub nowszej może łączyć się w trybie transportu CLIENT lub BINDINGS z dowolnym menedżerem kolejek produktu IBM MQ w usłudze.

Korzystanie z adaptera zasobów z produktem WebSphere Liberty

Aby połączyć się z IBM MQ z WebSphere Liberty, należy użyć adaptera zasobów IBM MQ . Ponieważ produkt Liberty nie zawiera adaptera zasobów produktu IBM MQ , należy go uzyskać oddzielnie od serwisu Fix Central. Wersja adaptera zasobów zależy od wersji Java EE serwera aplikacji.

Więcej informacji na temat pobierania i instalowania adaptera zasobów zawiera sekcja “Instalowanie adaptera zasobów w produkcie Liberty” na stronie 452.

Pojęcia pokrewne

“Konfigurowanie adaptera zasobów na potrzeby komunikacji przychodzącej” na stronie 458
Aby skonfigurować komunikację przychodzącą, należy zdefiniować właściwości jednego lub większej liczby obiektów ActivationSpec .

“Konfigurowanie adaptera zasobów na potrzeby komunikacji wychodzącej” na stronie 477
Aby skonfigurować komunikację wychodzącą, należy zdefiniować właściwości obiektu ConnectionFactory i administrowanego obiektu docelowego.

“użycieIBM MQ classes for JMS” na stronie 83
IBM MQ classes for Java Message Service (IBM MQ classes for JMS) jest dostawcą JMS dostarczonym z produktem IBM MQ. Oprócz implementowania interfejsów zdefiniowanych w pakiecie javax.jms produkt IBM MQ classes for JMS udostępnia dwa zestawy rozszerzeń do interfejsu API produktu JMS .

“użycieIBM MQ classes for Java” na stronie 343
Użyj produktu IBM MQ w środowisku Java . Program IBM MQ classes for Java umożliwia aplikacji Java łączenie się z serwerem IBM MQ jako klient IBM MQ lub bezpośrednie połączenie z menedżerem kolejek produktu IBM MQ .

Zadania pokrewne

Konfigurowanie serwera aplikacji pod względem używania najnowszego poziomu konserwacyjnego adaptera zasobów

Określanie problemu dla adaptera zasobów produktu IBM MQ

WebSphere Application Server tematy

Konserwowanie adaptera zasobów produktu IBM MQ

Wdrażanie aplikacji produktu JMS na serwerze Liberty w celu korzystania z dostawcy przesyłania komunikatów produktu IBM MQ

Obsługa instrukcji adaptera zasobów IBM MQ

Adapter zasobów dostarczany z produktem IBM MQ 8.0 lub nowszym 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 w związku z tym obsługuje produkt JMS 2.0.

Lista certyfikowanych serwerów aplikacji jest obsługiwana w systemie Serwis WWW produktu Oracle.

Wdrażanie w produkcie WebSphere Liberty

WebSphere Liberty 8.5.5 Fix Pack 6 i nowsze, WebSphere Application Server Liberty 9.0 i nowsze są certyfikowanymi serwerami aplikacji Java EE 7 , dzięki czemu można w nich wdrażać adapter zasobów produktu IBM MQ 9.0 .

Produkt WebSphere Liberty udostępnia dwie funkcje do pracy z adapterami zasobów:

- Opcja wmqJmsClient-1.1 umożliwia pracę z adapterami zasobów produktu JMS 1.1 .
- Opcja wmqJmsClient-2.0 umożliwia pracę z adapterami zasobów produktu JMS 2.0 .

Ważne: Adapter zasobów produktu IBM MQ 8.0 lub nowszy musi zostać wdrożony za pomocą funkcji wmqJmsClient-2.0 .

Informacje na temat tej konfiguracji znajdują się w scenariuszu Podłączanie serwera WebSphere Liberty Liberty do produktu IBM MQ.

Wdrażanie w produkcie WebSphere Application Server traditional

Produkt WebSphere Application Server traditional 9.0 jest dostarczany z adapterem zasobów produktu IBM MQ 9.0 , który jest już zainstalowany. Z tego powodu nie ma potrzeby instalowania nowego adaptera zasobów. Zainstalowany adapter zasobów może łączyć się w trybie transportu CLIENT lub BINDINGS z dowolnymi menedżerami kolejek, które działają w obsługiwanej wersji produktu IBM MQ lub IBM WebSphere MQ. Aby uzyskać więcej informacji, zapoznaj się z sekcją: “Połączenia z menedżerami kolejek produktu IBM MQ 8.0 lub nowszego” na stronie 445.

Ważne: Adapter zasobów produktu IBM MQ 9.0 nie może zostać wdrożony w wersjach produktu WebSphere Application Server traditional przed produktem IBM MQ 9.0, ponieważ te wersje nie są certyfikowane przez produkt Java EE 7 .

Dowolna obsługiwana wersja produktu WebSphere Application Server może używać adaptera zasobów produktu IBM MQ , który jest dołączony do tego adaptera w celu nawiązania połączenia z dowolną obsługiwaną wersją produktu IBM MQ.

Korzystanie z adaptera zasobów z innymi serwerami aplikacji

W przypadku wszystkich innych serwerów aplikacji zgodnych z produktem Java EE 7 , problemy występujące po pomyślnym zakończeniu testu weryfikowania instalacji (IVT) adaptera zasobów produktu IBM MQ mogą zostać zgłoszone do produktu IBM w celu zbadania śledzenia produktu IBM MQ i innych informacji diagnostycznych produktu IBM MQ . Jeśli program IVT adaptera zasobów produktu IBM MQ nie może zostać uruchomiony pomyślnie, napotkane problemy mogą zostać spowodowane przez niepoprawne wdrożenie lub niepoprawne definicje zasobów, które są specyficzne dla serwera aplikacji, a problemy muszą zostać zbadane przy użyciu dokumentacji serwera aplikacji i/lub organizacji wsparcia dla tego serwera aplikacji.

Java Środowisko wykonawcze

Środowisko wykonawcze programu Java (JRE) używane do uruchamiania serwera aplikacji musi być obsługiwane przez produkt IBM MQ 9.0 lub nowszy. Więcej informacji na ten tematzawiera sekcja Wymagania systemowe produktu IBM MQ. (Wybierz wersję i raport systemu operacyjnego lub komponentu, który chcesz zobaczyć, a następnie kliknij odsyłacz **Java** , który znajduje się na karcie **Obsługiwane oprogramowanie**).

Połączenia z menedżerami kolejek produktu IBM MQ 8.0 lub nowszego

Pełny zakres funkcji produktu JMS 2.0 jest dostępny podczas nawiązywania połączenia z menedżerem kolejek w wersji IBM MQ 8.0 lub nowszej za pomocą adaptera zasobów wdrożonego na certyfikowanym serwerze aplikacji produktu Java EE 7 . Więcej informacji na temat wersji adaptera zasobów, które są dostarczane z produktem WebSphere Application Server, zawiera nota techniczna Która wersja adaptera zasobów WebSphere MQ (RA) jest dostarczana z serwerem WebSphere Application Server?.

Aby można było korzystać z funkcji produktu JMS 2.0 , adapter zasobów musi nawiązać połączenie z menedżerem kolejek przy użyciu trybu normalnego dostawcy przesyłania komunikatów produktu IBM MQ . Więcej informacji na ten temat zawiera sekcja Konfigurowanie właściwości JMS **PROVIDERVERSION**.

Połączenia z menedżerami kolejek produktu IBM WebSphere MQ 7.5 lub wcześniejszymi

Jest on obsługiwany w celu wdrożenia adaptera zasobów w wersji IBM MQ 9.0 lub nowszej na certyfikowanym serwerze aplikacji Java EE 7 , który obsługuje produkt JMS 2.0 i łączy się z tym adapterem zasobów z menedżerem kolejek, na którym działa produkt IBM WebSphere MQ 7.5 lub wcześniejszy. Dostępne funkcje są ograniczone możliwościami menedżera kolejek. Więcej informacji na ten temat zawiera sekcja Konfigurowanie właściwości JMS **PROVIDERVERSION**.

Rozszerzenia MQ

Specyfikacja JMS 2.0 wprowadza zmiany dotyczące sposobu pracy niektórych zachowań. Ponieważ produkt IBM MQ 8.0 lub nowszy implementuje tę specyfikację, istnieją zmiany w zachowaniu między produktem IBM MQ 8.0 a nowszymi i wersjami przed produktem IBM MQ 8.0. W produkcie IBM MQ 8.0 i nowszych produkt IBM MQ classes for JMS obejmuje obsługę właściwości systemowej Java `com.ibm.mq.jms.SupportMQExtensions` , która po ustawieniu wartości TRUEpowoduje, że te wersje produktu IBM MQ będą przywracać te zachowania w produkcie IBM WebSphere MQ 7.5 lub starszym. 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 `com.ibm.mq.jms.SupportMQExtensions` Java . Ta właściwość adaptera zasobów jest domyślnie ustawiona na wartość false w polu `ra.xml` .

Jeśli ustawiona jest zarówno właściwość adaptera zasobów, jak i właściwość systemowa Java , to właściwość systemowa ma pierwszeństwo.

Należy zauważyć, że w ramach adaptera zasobów, który jest już wdrożony w produkcie WebSphere Application Server traditional 9.0, ta właściwość jest automatycznie ustawiana na wartość TRUE , aby pomóc migracji.

Więcej informacji na ten temat zawiera “Właściwość SupportMQExtensions” na stronie 324.

Zagadnienia ogólne

Interwyjeżdżanie sesji nie jest obsługiwane

Niektóre serwery aplikacji udostępniają funkcję o nazwie interwyjeżdżanie sesji, w której ta sama sesja produktu JMS może być używana w wielu transakcjach, mimo że jest ona wyświetlana tylko w jednym czasie. Adapter zasobów produktu IBM MQ nie obsługuje tej możliwości, co może prowadzić do następujących problemów:

```
Próba umieszczenia komunikatu w kolejce produktu MQ nie powiodła się. Kod przyczyny: 2072
(MQRC_SYNCPOINT_NOT_AVAILABLE).
```

Wywołania funkcji xa_close () kończą się niepowodzeniem z kodem przyczyny -3 (XAER_PROTO), a kod FDC o identyfikatorze sondy AT040010 jest generowany w menedżerze kolejek produktu IBM MQ , do którego uzyskiwany jest dostęp z serwera aplikacji. Informacje na temat wyłączania tej możliwości można znaleźć w dokumentacji serwera aplikacji.

Java Transaction API (JTA)-specyfikacja sposobu odtwarzania zasobów XA na potrzeby odtwarzania transakcji XA.

Sekcja 3.4.8 specyfikacji JTA nie definiuje konkretnego mechanizmu, za pomocą którego zasoby XA są ponownie tworzone w celu wykonania operacji odtwarzania transakcyjnego XA. W związku z tym należy do każdego menedżera transakcji (a w związku z tym serwera aplikacji), w jaki sposób zasoby XA związane z transakcją XA są odtwarzane. Możliwe jest, że w przypadku niektórych serwerów aplikacji adapter zasobów produktu IBM MQ 9.0 nie implementuje mechanizmów specyficznych dla serwera aplikacji, które są używane do wykonywania operacji odtwarzania transakcyjnego XA.

Uzgadniaj połączenia w fabryce ManagedConnection

Serwer aplikacji może wywołać metodę matchManagedConnections w instancji fabryki ManagedConnectionudostępnianej przez adapter zasobów produktu IBM MQ . Obiekt ManagedConnection jest zwracany tylko wtedy, gdy metoda znajdzie jeden, który jest zgodny z argumentami **`javax.security.auth.Subject`** i **`javax.resource.spi.ConnectionRequestInfo`** , które zostały przekazane do metody przez serwer aplikacji.

Ograniczenia adaptera zasobów produktu IBM MQ

Adapter zasobów produktu IBM MQ jest obsługiwany na wszystkich platformach IBM MQ . Jednak w przypadku korzystania z adaptera zasobów IBM MQ niektóre funkcje produktu IBM MQ są niedostępne lub ograniczone.

Adapter zasobów produktu IBM MQ ma następujące ograniczenia:

- Od wersji IBM MQ 8.0adapter zasobów jest adapterem zasobów Java Platform, Enterprise Edition 7 (Java EE 7) udostępniający funkcję JMS 2.0 . W związku z tym adapter zasobów IBM MQ 8.0 lub nowszy musi być zainstalowany na certyfikowanym serwerze aplikacji w wersji Java EE 7 lub nowszej. Może on łączyć się w trybie transportu klienta lub powiązań z dowolnym menedżerem kolejek w usłudze.
- W przypadku działania wewnątrz serwera aplikacji WebSphere Liberty stabilizacja produktu IBM MQ classes for Java nie jest obsługiwana. W innych serwerach aplikacji nie zaleca się używania produktu IBM MQ classes for Java . See the IBM technote Korzystanie z interfejsów Java produktu WebSphere MQ w środowiskach J2EE/JEE for details of IBM MQ classes for Java considerations within Java EE.
- W przypadku działania wewnątrz serwera aplikacji WebSphere Liberty w systemie z/OSnależy użyć opcji wmqJmsClient-2.0 . Ogólne wsparcie dla produktu JCA nie jest możliwe dla produktu z/OS.
- Adapter zasobów produktu IBM MQ nie obsługuje programów obsługi wyjścia kanałów, które są napisane w językach innych niż Java.
- Gdy serwer aplikacji jest uruchomiony, wartość wymaganej właściwości sslFipsmusi mieć wartość true dla wszystkich zasobów JCA lub wartości false dla wszystkich zasobów JCA . Jest to wymaganie nawet wtedy, gdy zasoby produktu JCA nie są używane współbieżnie. Jeśli wymagana właściwość sslFipsma inne wartości dla różnych zasobów produktu JCA , produkt IBM MQ wydaje kod przyczyny MQRC_UNSUPPORTED_CIPHER_SUITE, nawet jeśli połączenie TLS nie jest używane.
- Nie można określić więcej niż jednego magazynu kluczy dla serwera aplikacji. Jeśli połączenia są nawiązane do więcej niż jednego menedżera kolejek, wszystkie połączenia muszą korzystać z tego samego magazynu kluczy. To ograniczenie nie ma zastosowania do produktu WebSphere Application Server.
- Jeśli używana jest tabela definicji kanału klienta (CCDT) z więcej niż jedną odpowiednią definicją kanału połączenia klienta, w przypadku awarii adapter zasobów może wybrać inną definicję kanału i w związku z tym inny menedżer kolejek z tabeli definicji kanału klienta, co powodowałoby problemy podczas odtwarzania transakcji. Adapter zasobów nie podejmuje żadnych działań w celu uniknięcia użycia takiej konfiguracji, a użytkownik jest odpowiedzialny za unikanie konfiguracji, które mogą powodować problemy związane z odtwarzaniem transakcji.
- Funkcje ponawiania połączenia wprowadzone w produkcie IBM WebSphere MQ 7.0.1 nie są obsługiwane w przypadku połączeń wychodzących podczas działania w kontenerze Java EE (EJB/Servlet). Ponowienie połączenia nie jest obsługiwane w przypadku wszystkich wychodzących JMS , gdy adapter jest używany w kontekście kontenera produktu JEE , niezależnie od konfiguracji transakcji lub w przypadku użycia nietransakcyjnego.
- Ponowne uwierzytelnianie, zgodnie z definicją w sekcji 9.1.9 specyfikacji produktu Java EE Connector Architecture w wersji 1.7 , nie jest obsługiwane przez produkt JMS . Plik `ra.xml` w adpaterze zasobów IBM MQ musi mieć właściwość o nazwie **`reauthentication-support`** ustawioną na wartość `false`. Próba ponownego uwierzytelnienia przez serwer aplikacji wyników połączenia JMS w adapterze zasobów IBM MQ zgłaszająca wyjątek javax.resource.spi.SecurityException z kodem komunikatu MQJCA1028 .

Zadania pokrewne

Określanie, że w czasie wykonywania w kliencie MQI są używane tylko specyfikacje CipherSpecs z certyfikatem FIPS

Odsyłacze pokrewne

FIPS (Federal Information Processing Standards) dla produktów UNIX, Linux i Windows

WebSphere Application Server i adapter zasobów IBM MQ

Adapter zasobów produktu IBM MQ jest używany przez aplikacje, które wykonują przesyłanie komunikatów produktu JMS z dostawcą przesyłania komunikatów produktu IBM MQ w produkcie WebSphere Application Server.

Ważne: Nie należy używać adaptera zasobów IBM MQ ani IBM WebSphere MQ z produktem WebSphere Application Server 6.0 lub WebSphere Application Server 6.1.

Produkty WebSphere Application Server 7.0 i WebSphere Application Server 8.0 zawierają wersję adaptera zasobów produktu IBM WebSphere MQ 7.0 .

Produkt WebSphere Application Server 8.5.5 zawiera wersję adaptera zasobów produktu IBM WebSphere MQ 7.1 .

Program WebSphere Application Server traditional 9.0 zawiera wersję adaptera zasobów IBM MQ 9.0 . Adaptera zasobów w wersji IBM MQ 9.0 lub nowszej nie można wdrożyć we wcześniejszych wersjach produktu WebSphere Application Server, ponieważ te wersje nie są certyfikowane przez produkt Java EE 7 .

Jeśli do uzyskiwania dostępu do zasobów menedżera kolejek produktu IBM MQ z poziomu produktu WebSphere Application Serverma być używana aplikacja JMS , należy użyć dostawcy przesyłania komunikatów produktu IBM MQ w produkcie WebSphere Application Server. Dostawca przesyłania komunikatów produktu IBM MQ zawiera wersję produktu IBM MQ classes for JMS. Więcej informacji na ten temat zawiera nota techniczna Która wersja adaptera zasobów WebSphere MQ (RA) jest dostarczana z produktem WebSphere Application Server?.

Ważne: Nie należy uwzględniać żadnego z plików JAR produktu IBM MQ classes for JMS ani IBM MQ classes for Java w aplikacji. Może to spowodować wystąpienie wyjątków ClassCasti może być trudne do utrzymania.

Liberty i adapter zasobów IBM MQ

Adapter zasobów produktu IBM MQ może zostać zainstalowany w produkcie WebSphere Liberty 8.5.5 Fix Pack 2 lub nowszym za pomocą funkcji wmqJmsClient-1.1 lub wmqJmsClient-2.0 , w zależności od wersji instalowanego adaptera zasobów. Ewentualnie można, z pewnymi ograniczeniami, zainstalować adapter zasobów przy użyciu ogólnej obsługi produktu Java Platform, Enterprise Edition Connector Architecture (Java EE JCA).

Ogólne ograniczenia dotyczące instalowania adaptera zasobów w produkcie Liberty

Następujące ograniczenia dotyczą adaptera zasobów w przypadku używania funkcji wmqJmsClient-1.1 lub wmqJmsClient-2.0 , a także w przypadku korzystania z ogólnej obsługi produktu JCA :

- IBM MQ classes for Java nie są obsługiwane w produkcie Liberty. Nie mogą być one używane zarówno z funkcją przesyłania komunikatów IBM MQ Liberty , jak i z ogólną obsługą produktu JCA . Więcej informacji na ten temat zawiera sekcja Korzystanie z interfejsów Java produktu WebSphere MQ 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 .
- Przed IBM MQ 9.0opcja Advanced Message Security (AMS) nie została włączona do funkcji przesyłania komunikatów IBM MQ Liberty . Jednak produkt AMS jest obsługiwany w przypadku adaptera zasobów w wersji IBM MQ 9.0 lub nowszej.

Ograniczenia dotyczące korzystania z opcji produktu Liberty

W przypadku produktu WebSphere Liberty 8.5.5 Fix Pack 2 do produktu WebSphere Liberty 8.5.5 Fix Pack 5 włącznie, dostępna była tylko funkcja wmqJmsClient-1.1 i można było użyć tylko produktu JMS 1.1 . Produkt WebSphere Liberty 8.5.5 Fix Pack 6 dodał opcję wmqJmsClient-2.0 , aby można było użyć produktu JMS 2.0 .

Jednak funkcja, której należy używać, zależy od wersji adaptera zasobów, który jest używany:

- Adapter zasobów IBM WebSphere MQ 7.5.0 Fix Pack 6 i nowsze IBM WebSphere MQ 7.5 może być używany tylko z opcją wmqJmsClient-1.1 .
- Adapter zasobów IBM MQ 8.0.0 Fix Pack 3 i nowsze IBM MQ 8.0 może być używany tylko z opcją wmqJmsClient-2.0 .
- Adapter zasobów produktu IBM MQ 9.0 może być używany tylko z opcją wmqJmsClient-2.0 .

Ograniczenia dotyczące korzystania z ogólnej obsługi produktu JCA

Jeśli używana jest ogólna obsługa produktu JCA , zastosowanie mają następujące ograniczenia:

- Poziom produktu JMS należy określić podczas korzystania z ogólnej obsługi produktu JCA :
  - Produkty JMS 1.1 i JCA 1.6 muszą być używane tylko z adapterami zasobów IBM WebSphere MQ 7.5.0 Fix Pack 6 i nowsze IBM WebSphere MQ 7.5 .
  - Produkty JMS 2.0 i JCA 1.7 muszą być używane tylko z adapterami zasobów IBM MQ 8.0.0 Fix Pack 3 i nowsze IBM MQ 8.0 .
- Nie jest możliwe uruchomienie adaptera zasobów IBM MQ w systemie z/OS przy użyciu ogólnej obsługi produktu JCA . Aby uruchomić adapter zasobów produktu IBM MQ w systemie z/OS, musi on zostać uruchomiony z opcją wmqJmsClient-1.1 lub wmqJmsClient-2.0 .
- Położenie adaptera zasobów jest określane przy użyciu następującego elementu XML:

```
<resourceAdapter id="mqJms" location="${server.config.dir}/wmq.jmsra.rar">
  <classloader apiTypeVisibility="spec, ibm-api, api, third-party"/>
</resourceAdapter>
```

Ważne: Wartość znacznika identyfikatora może być dowolna z opcji EXCEPT dla produktu `wmqJms`. Jeśli jako identyfikator używany jest produkt `wmqJms` , wówczas produkt Liberty nie będzie mógł poprawnie załadować adaptera zasobów. Jest to spowodowane tym, że `wmqJms` jest identyfikatorem używanym wewnętrznie do odwołania się do konkretnej funkcji produktu IBM MQ. W rzeczywistości jest tworzony wyjątek NullPointer.

W poniższych przykładach przedstawiono niektóre fragmenty kodu z pliku `server.xml` :

```
<!-- 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 funkcji jca-1.7 i jms-2.0 oraz braku funkcji 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 produktu `mqJms` dla identyfikatora, który jest preferowany. Nie należy używać produktu `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 pomocą identyfikatora `mqJms`. Jest to wymagane, aby umożliwić załadowanie klas specyficznych dla produktu IBM MQ .

Ograniczenia dotyczące śledzenia przy użyciu ogólnej obsługi produktu JCA

Śledzenie i rejestrowanie nie są zintegrowane w systemie śledzenia Liberty . Zamiast tego śledzenie adaptera zasobów produktu IBM MQ musi być włączone przy użyciu właściwości systemowych produktu Java lub pliku konfiguracyjnego produktu IBM MQ classes for JMS , zgodnie z opisem w sekcji Śledzenie klas IBM MQ dla aplikacji JMS. Szczegółowe informacje na temat ustawiania właściwości systemowych produktu Java w produkcie Libertymożna znaleźć w dokumentacji produktuWebSphere Liberty.

Na przykład, aby włączyć śledzenie adaptera zasobów produktu IBM MQ w produkcie Liberty 19.0.0.9, należy dodać wpis do pliku Liberty `jvm.options`:

1. Utwórz plik tekstowy o nazwie `jvm.options`.
2. Wstaw następujące opcje maszyny JVM, aby włączyć śledzenie, po jednej w wierszu, w tym pliku:

```
-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 do pojedynczego serwera, zapisz `jvm.options` pod adresem:

   ```
   ${server.config.dir}/jvm.options
   ```

   Aby zastosować te zmiany do wszystkich Liberty, zapisz `jvm.options` pod adresem:

   ```
   ${wlp.install.dir}/etc/jvm.options
   ```

   Ta opcja będzie obowiązywać dla wszystkich maszyn JVM, które nie mają lokalnie zdefiniowanego pliku `jvm.options` .
4. Zrestartuj serwer, aby włączyć zmiany.

Wyniki śledzenia są zapisywane w pliku śledzenia o nazwie `MQRA-WLP_<process identifier>.trc` w katalogu `<path_to_trace_to>`.

Pełna obsługa interfejsu Liberty XA przy użyciu tabel definicji kanału klienta

V 9.1.2

W przypadku korzystania z produktu WebSphere Liberty 18.0.0.2 , przy użyciu produktu IBM MQ 9.1.2, można korzystać z grup menedżerów kolejek w tabeli definicji kanału klienta (CCDT) w połączeniu z transakcjami XA. Oznacza to, że obecnie możliwe jest korzystanie z dystrybucji i dostępności obciążeń, udostępnianych przez grupy menedżerów kolejek, przy jednoczesnym zachowaniu integralności transakcji.

W przypadku wystąpienia błędów połączenia z menedżerem kolejek, menedżer kolejek musi zostać ponownie dostępny, aby możliwa była rozstrzyganie transakcji. Odtwarzanie transakcji jest zarządzane przez produkt Libertyi może być konieczne skonfigurowanie menedżera transakcji, tak aby możliwe było ponowne udostępnienie menedżerów kolejek przez odpowiedni okres. Więcej informacji na ten temat zawiera sekcja Menedżer transakcji (transakcja) w dokumentacji produktu WebSphere Liberty .

Jest to opcja po stronie klienta, oznacza to, że potrzebny jest adapter zasobów IBM MQ 9.1.2 , a nie menedżer kolejek produktu IBM MQ 9.1.2 .

Instalowanie adaptera zasobów produktu IBM MQ

Adapter zasobów produktu IBM MQ jest dostarczany jako plik archiwum zasobów (Resource Archive-RAR). Zainstaluj plik RAR na serwerze aplikacji. Może być konieczne dodanie katalogów do ścieżki systemowej.

O tym zadaniu

Adapter zasobów produktu IBM MQ jest dostarczany jako plik archiwum zasobów (Resource Archive-RAR) o nazwie `wmq.jmsra.rar`. Plik RAR zawiera IBM MQ classes for JMS i IBM MQ implementację interfejsów Java EE Connector Architecture (JCA).

Podczas instalowania adaptera zasobów w ramach instalacji produktu IBM MQ produkt `wmq.jmsra.rar` jest instalowany z produktem IBM MQ classes for JMS w katalogu podanym w sekcji Tabela 60 na stronie 450.

Tabela 60. Katalog, w którym znajduje się plik wmq.jmsra.rar dla każdej platformy

Platforma	**Katalog**
UNIX and Linux	*MQ_INSTALLATION_PATH*/java/lib/jca
IBM i	/QIBM/ProdData/mqm/java/lib/jca
Windows	*MQ_INSTALLATION_PATH*\java\lib\jca

Tabela 60. Katalog, w którym znajduje się plik wmq.jmsra.rar dla każdej platformy (kontynuacja)	
Platforma	**Katalog**
z/OS	*MQ_INSTALLATION_PATH*/java/lib/jca

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Aby nawiązać połączenie z serwerem IBM MQ z serwera aplikacji, należy użyć adaptera zasobów produktu IBM MQ . W zależności od tego, który serwer aplikacji jest używany, adapter zasobów może być wstępnie zainstalowany lub może być konieczne zainstalowanie go samodzielnie.

Tabela 61. Instalacja adaptera zasobów na serwerze aplikacji	
Serwer aplikacji	**Instalacja wstępnie zainstalowana lub wymagana?**
WebSphere Application Server traditional 9.0	Adapter zasobów produktu IBM MQ 9.0 jest wstępnie zainstalowany w produkcie WebSphere Application Server traditional 9.0. Z tego powodu nie ma potrzeby instalowania nowego adaptera zasobów w produkcie WebSphere Application Server traditional 9.0.
WebSphere Liberty	Produkt WebSphere Liberty nie zawiera adaptera zasobów produktu IBM MQ , dlatego należy go uzyskać oddzielnie od serwisu Fix Central.
Inny serwer aplikacji Java EE	Adapter zasobów należy uzyskać oddzielnie od produktu Fix Central, tak jak w przypadku produktu WebSphere Liberty.

Procedura

- Jeśli nawiążesz połączenie z produktem IBM MQ z produktu WebSphere Libertylub innego serwera aplikacji Java EE , pobierz i zainstaluj adapter zasobów produktu IBM MQ zgodnie z opisem w sekcji “Instalowanie adaptera zasobów w produkcie Liberty” na stronie 452.
- Linux UNIX

  W przypadku połączeń powiązań w systemach UNIX and Linux należy sprawdzić, czy katalog zawierający biblioteki JNI (Native Interface) produktu Java znajduje się w ścieżce systemowej.

  Informacje na temat położenia tego katalogu, w którym znajdują się również biblioteki produktu IBM MQ classes for JMS , zawiera sekcja “Konfigurowanie bibliotek rodzimego interfejsu produktu Java (JNI)” na stronie 93.

  Windows W systemie Windowsten katalog jest automatycznie dodawany do ścieżki systemowej podczas instalacji produktu IBM MQ classes for JMS.

  Wskazówka: Jako alternatywę dla ustawienia ścieżki systemowej, adapter zasobów IBM MQ ma właściwość o nazwie nativeLibraryŚcieżka, która może być używana do określania położenia biblioteki JNI. Na przykład w produkcie WebSphere Liberty zostanie to skonfigurowane tak, jak pokazano 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 produkcie Liberty

Aby nawiązać połączenie z produktem IBM MQ z serwera WebSphere Libertylub z innych serwerów aplikacji Java EE , należy użyć adaptera zasobów produktu IBM MQ . Ponieważ produkt Liberty nie zawiera adaptera zasobów produktu IBM MQ , należy go uzyskać oddzielnie od serwisu Fix Central.

Zanim rozpoczniesz

Uwaga: Informacje zawarte w tym temacie nie mają zastosowania do produktu WebSphere Application Server traditional 9.0. Adapter zasobów produktu IBM MQ 9.0 jest wstępnie zainstalowany w produkcie WebSphere Application Server traditional 9.0. Dlatego w tym przypadku nie jest wymagane zainstalowanie nowego adaptera zasobów.

Przed rozpoczęciem tej czynności należy upewnić się, że na komputerze jest zainstalowane środowisko Java runtime environment (JRE) oraz że środowisko JRE zostało dodane do ścieżki systemowej.

Instalator produktu Java , który jest używany w tym procesie instalacji, nie wymaga działania jako użytkownik root ani żaden konkretny użytkownik. Jedynym wymaganiem jest to, że użytkownik, który jest uruchamiany, ma dostęp do zapisu w katalogu, w którym mają być zapisywane pliki.

O tym zadaniu

Plik JAR dla adaptera zasobów, który można pobrać z programu Fix Central , jest wykonywalny. Po uruchomieniu tego pliku wykonywalnego wyświetlana jest umowa licencyjna produktu IBM MQ , która musi zostać zaakceptowana. Pyta on o katalog, w którym ma zostać zainstalowany adapter zasobów IBM MQ . Plik RAR adaptera zasobów i program testu weryfikacji instalacji (IVT) są następnie instalowane w tym katalogu. Można zaakceptować wartość domyślną lub określić inny katalog, który może być katalogiem adapterów zasobów serwera aplikacji, lub dowolnego innego katalogu w systemie. Katalog ten jest tworzony jako część instalacji, jeśli nie istnieje.

Przed IBM MQ 9.0nazwa pliku, który ma zostać pobrany, ma format *V.R.M.F*`-WS-MQ-Java-InstallRA.jar`, na przykład `8.0.0.6-WS-MQ-Java-InstallRA.jar`. W produkcie IBM MQ 9.0formatem nazwy pliku jest *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 programie WebSphere Liberty.

Procedura

1. Pobierz adapter zasobów IBM MQ z programu Fix Central.
   a) Kliknij ten odsyłacz: Adapter zasobów produktu IBM MQ.
   b) W wyświetlonej liście dostępnych poprawek znajdź adapter zasobów dla używanej wersji produktu IBM MQ .
      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 procesem pobierania.
2. Uruchom instalację, wprowadzając następującą komendę z katalogu, do którego pobrano plik.
   W programie IBM MQ 9.0format komendy jest następujący:

   ```
   java -jar V.R.M.F-IBM-MQ-Java-InstallRA.jar
   ```

   gdzie *V.R.M.F* to numer wersji, wydania, modyfikacji i pakietu poprawek, a *V.R.M.F*`-IBM-MQ-Java-InstallRA.jar` to nazwa pliku, który został pobrany z programu Fix Central.

Na przykład, aby zainstalować adapter zasobów IBM MQ dla wersji 9.1.4.0 , należy użyć następującej komendy:

```
java -jar 9.1.4.0-IBM-MQ-Java-InstallRA.jar
```

Uwaga: Aby przeprowadzić tę instalację, należy zainstalować na komputerze środowisko JRE i dodać je do ścieżki systemowej.

Po wprowadzeniu komendy wyświetlane są następujące informacje:

```
Zanim będzie można użyć, wyodrębnić lub zainstalować produkt IBM MQ V9.1, należy zaakceptować
warunki 1. IBM International License Agreement for Evaluation of
Programy 2. IBM International Program License Agreement i dodatkowe
informacje licencyjne. Przeczytaj uważnie poniższe umowy licencyjne.

Umowa licencyjna jest oddzielnie widziana przy użyciu
--viewLicenseOpcja umowy.
Naciśnij klawisz Enter, aby wyświetlić warunki licencji teraz, 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 zaraz po wybraniu x, pojawia się następujący komunikat, który informuje, że możesz wybrać opcję wyświetlania dodatkowych warunków licencji:

      ```
      Dodatkowe informacje o licencji są wyświetlane oddzielnie przy użyciu
      --viewLicense-opcja informacji.
      Naciśnij klawisz Enter, aby wyświetlić dodatkowe informacje licencyjne teraz, lub 'x', aby
      pominąć.
      ```

   b) Aby wyświetlić dodatkowe warunki licencji, naciśnij klawisz Enter.

      Naciśnięcie klawisza x powoduje również pominięcie wyświetlania dodatkowych warunków licencji.

      Po wyświetleniu dodatkowych warunków licencji lub bezpośrednio po wybraniu 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
      zgadzają się, wybierz "Nie zgadzam się".

      Wybierz [ 1] I Zgadzam się, lub [ 2] Nie zgadzam się:
      ```

   c) Aby zaakceptować umowę licencyjną i kontynuować wybieranie katalogu instalacyjnego, wybierz 1.

      Alternatywnie, jeśli zostanie wybrana opcja 2, instalacja zostanie zakończona natychmiast.

      Jeśli wybrano wartość 1, zostanie wyświetlony następujący komunikat z prośbą o wybranie docelowego katalogu instalacyjnego:

      ```
      Wprowadź katalog dla plików produktu lub pozostaw puste pole, aby zaakceptować wartość
      domyślną.
      Domyślnym katalogiem docelowym jest H: \Liberty\WMQ
      Katalog docelowy dla plików produktu?
      ```

4. Podaj katalog instalacyjny dla adaptera zasobów:

   - Jeśli chcesz zainstalować adapter zasobów w domyślnym położeniu, naciśnij klawisz Enter bez określania wartości.
   - Jeśli chcesz zainstalować adapter zasobów w innym miejscu niż domyślny, 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:

   ```
   Rozpakowywanie plików do H: \Liberty\WMQ \wmq
   Successfully extracted all product files.
   ```

   W trakcie instalacji w wybranym katalogu instalacyjnym tworzony jest nowy katalog o nazwie `wmq` , a następnie w katalogu `wmq` zostaną zainstalowane następujące pliki:

   - Program testowy weryfikacji instalacji, `wmq.jmsra.ivt`.
   - Plik RAR IBM MQ , `wmq.jmsra.rar`.

5. Skonfiguruj adapter zasobów w produkcie WebSphere Liberty.

   Kroki, które należy wykonać, aby skonfigurować adapter zasobów w programie Liberty , są następujące. 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 produktu IBM MQ 9.1 .

      Więcej informacji na ten temat zawiera “Która wersja adaptera zasobów jest używana” na stronie 443.

   b) Dodaj odwołanie do zainstalowanego pliku `wmq.jmsra.rar` .

   Uwaga: W przypadku wersji produktu Liberty do wersji WebSphere Liberty 8.5.5 Fix Pack 1, jeśli komponent EJB jest wdrażany przy użyciu wyłącznie konfiguracji w ramach produktu `ejb-jar.xml`, wersja produktu WebSphere Application Server , w której używany jest profil produktu Liberty , musi mieć zastosowanie poprawki APAR PM89890 . Ta metoda konfiguracji jest używana dla programu weryfikacji instalacji adaptera zasobów (IVT), dlatego ten raport APAR jest wymagany, aby program IVT został uruchomiony.

   Przykładowa konfiguracja do obsługi serwletów i komponentów MDB z produktem 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"/>
```

Konfigurowanie adaptera zasobów produktu IBM MQ

Aby skonfigurować adapter zasobów produktu IBM MQ , należy zdefiniować różne zasoby Java Platform, Enterprise Edition Connector Architecture (JCA) oraz, opcjonalnie, właściwości systemowe. Należy również skonfigurować adapter zasobów, aby uruchomić program test weryfikujący instalację (IVT). Jest to ważne, ponieważ usługa IBM może wymagać uruchomienia tego programu w celu wskazania, że dowolny serwer aplikacji inny niżIBM został poprawnie skonfigurowany.

Zanim rozpoczniesz

W przypadku tego zadania założono, że użytkownik zna już produkt JMS i IBM MQ classes for JMS. Wiele właściwości używanych do konfigurowania adaptera zasobów produktu IBM MQ jest odpowiednikiem właściwości obiektów produktu IBM MQ classes for JMS i ma tę samą funkcję.

O tym zadaniu

Każdy serwer aplikacji udostępnia własny zestaw interfejsów administracyjnych. Niektóre serwery aplikacji udostępniają graficzne interfejsy użytkownika służące do definiowania zasobów produktu JCA , ale inne wymagają, aby administrator zapisał plany wdrożenia XML. W związku z tym, poza zakresem tej dokumentacji, informacje na temat konfigurowania adaptera zasobów produktu IBM MQ dla każdego serwera aplikacji są dostępne.

W związku z tym poniższe kroki koncentrują się tylko na tym, co należy skonfigurować. Informacje na temat konfigurowania adaptera zasobów produktu JCA można znaleźć w dokumentacji dostarczonej wraz z serwerem aplikacji.

Procedura

Zdefiniuj zasoby produktu JCA w następujących kategoriach:

- Zdefiniuj właściwości obiektu ResourceAdapter .

Właściwości te, które reprezentują właściwości globalne adaptera zasobów, takie jak poziom śledzenia diagnostycznego, są opisane w sekcji “Konfiguracja właściwości obiektu ResourceAdapter” na stronie 456.

- Zdefiniuj właściwości obiektu ActivationSpec .

  Te właściwości określają, w jaki sposób komponent MDB jest aktywowany na potrzeby komunikacji przychodzącej. Więcej informacji na ten temat zawiera “Konfigurowanie adaptera zasobów na potrzeby komunikacji przychodzącej” na stronie 458.

- Zdefiniuj właściwości obiektu ConnectionFactory .

  Serwer aplikacji korzysta z tych właściwości w celu utworzenia obiektu JMS ConnectionFactory na potrzeby komunikacji wychodzącej. Więcej informacji na ten temat zawiera sekcja “Konfigurowanie adaptera zasobów na potrzeby komunikacji wychodzącej” na stronie 477.

- Zdefiniuj właściwości administrowanego obiektu docelowego.

  Serwer aplikacji korzysta z tych właściwości w celu 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 477.

- Opcjonalne: Zdefiniuj plan wdrożenia dla adaptera zasobów.

  Plik RAR adaptera zasobów produktu 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 produkcie https://xmlns.jcp.org/xml/ns/javaee/connector_1_7.xsd i zawiera informacje na temat adaptera zasobów i udostępnianych przez niego usług. Serwer aplikacji może również wymagać planu wdrożenia dla adaptera zasobów. Ten plan wdrożenia jest specyficzny dla serwera aplikacji.

Określ właściwości systemowe maszyny JVM zgodnie z wymaganiami:

- Jeśli używany jest protokół TLS (Transport Layer Security), określ położenia pliku kluczy i pliku zaufanych certyfikatów jako właściwości systemu JVM, tak jak w następującym 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 ani obiektu ConnectionFactory i nie można określić więcej niż jednego magazynu kluczy dla serwera aplikacji. Właściwości mają zastosowanie do całej maszyny JVM, a zatem mogą mieć wpływ na serwer aplikacji, jeśli inne aplikacje działające na serwerze aplikacji korzystają z połączeń TLS. Serwer aplikacji może również resetować 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 242.

- Opcjonalne: Jeśli jest to wymagane, skonfiguruj adapter zasobów w taki sposób, aby protokołował komunikaty ostrzegawcze do standardowego dziennika wyjściowego 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 w produkcie IBM MQ classes for JMS. Oznacza to, że domyślnie komunikaty są wysyłane do pliku o nazwie `mqjms.log`. Aby skonfigurować adapter zasobów w celu dodatkowego rejestrowania komunikatów ostrzegawczych w standardowym dzienniku wyjściowym serwera aplikacji, ustaw następującą właściwość systemową maszyny JVM dla serwera aplikacji:

  ```
  -Dcom.ibm.msg.client.commonservices.log.outputName=mqjms.log,stdout
  ```

  Jest to ta sama właściwość, jak ta, która jest używana do sterowania śledzeniem dla IBM MQ classes for JMS. Podobnie jak w przypadku produktu IBM MQ classes for JMS, możliwe jest użycie właściwości systemowej wskazującego plik `jms.config` (patrz sekcja “Plik konfiguracyjny IBM MQ classes for JMS” na stronie 95). Informacje na temat ustawiania właściwości systemowej maszyny JVM można znaleźć w dokumentacji serwera aplikacji.

Skonfiguruj adapter zasobów, aby uruchomić test weryfikujący instalację

- Skonfiguruj adapter zasobów, aby uruchomić program test weryfikacji instalacji (IVT) dostarczony z adapterem zasobów produktu IBM MQ .

  Informacje na temat tego, co należy skonfigurować w celu uruchomienia programu IVT, zawiera sekcja "Weryfikowanie instalacji adaptera zasobów" na stronie 498.

  Jest to ważne, ponieważ usługa IBM może wymagać uruchomienia tego programu w celu wskazania, że dowolny serwer aplikacji inny niżIBM został poprawnie skonfigurowany.

  Ważne: Przed uruchomieniem programu należy skonfigurować adapter zasobów.

Konfiguracja właściwości obiektu ResourceAdapter

Obiekt ResourceAdapter hermetykuje globalne właściwości adaptera zasobów produktu IBM MQ , takie jak poziom śledzenia diagnostycznego. Aby zdefiniować te właściwości, należy skorzystać z narzędzi adaptera zasobów, zgodnie z opisem podanym w dokumentacji dostarczonej wraz z serwerem aplikacji.

Obiekt ResourceAdapter ma dwa zestawy właściwości:

- Właściwości powiązane z śledzeniem diagnostycznym
- Właściwości powiązane z pulą połączeń zarządzaną przez adapter zasobów

Sposób definiowania tych właściwości zależy od interfejsów administracyjnych udostępnianych przez serwer aplikacji. Jeśli używany jest produkt WebSphere Application Server traditional, patrz "WebSphere Application Server traditional Konfiguracja" na stronie 458 lub jeśli używany jest produkt WebSphere Liberty, patrz sekcja "WebSphere Liberty Konfiguracja" na stronie 458. W przypadku innych serwerów aplikacji należy zapoznać się z dokumentacją produktu dla serwera aplikacji.

Więcej informacji na temat definiowania właściwości powiązanych ze śledzeniem diagnostycznym zawiera sekcja Śledzenie adaptera zasobów produktu IBM MQ .

Adapter zasobów zarządza wewnętrzną pulą połączeń programu JMS , które są używane do dostarczania komunikatów do baz danych MDB. Tabela 62 na stronie 456 zawiera listę właściwości obiektu ResourceAdapter , które są powiązane z pulą połączeń.

Tabela 62. Właściwości obiektu ResourceAdapter , które są 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 mają współużytkować połączenie JMS . Współużytkowanie połączeń nie jest możliwe, a ta właściwość zawsze ma wartość 1.
Liczba reconnectionRetry	Łańcuch	5	Maksymalna liczba prób nawiązania połączenia przez adapter zasobów z menedżerem kolejek produktu IBM MQ , jeśli nawiązanie połączenia nie powiedzie się.
Odstęp czasu reconnectionRetry	Łańcuch	300 000	Mierzony w milisekundach czas, przez jaki adapter zasobów czeka 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, gdy serwer aplikacji jest uruchomiony.

Tabela 62. Właściwości obiektu ResourceAdapter , które są powiązane z pulą połączeń (kontynuacja)			
Nazwa właściwości	**Typ**	**Wartość domyślna**	**Opis**
Przedział czasu startupRetry	Łańcuch	30 000	Domyślny czas uśpienia między kolejnymi próbami nawiązania połączenia startowego (w milisekundach).
supportMQExtensions	Łańcuch	Fałsz	Powoduje cofanie zachowania IBM MQ JMS do zachowania sprzedJMS 2.0 . Więcej informacji na ten temat zawiera sekcja "Właściwość SupportMQExtensions" na stronie 324.
nativeLibraryŚcieżka	Łańcuch	<puste>	Ścieżka, która ma być używana do ładowania biblioteki JNI produktu IBM MQ w celu umożliwienia połączenia w trybie powiązań. Windows W systemie Windows ścieżka systemowa musi również zawierać położenie zgodnej instalacji produktu IBM MQ .

Gdy komponent MDB jest wdrożony na serwerze aplikacji, tworzone jest nowe połączenie produktu JMS , a konwersacja uruchomiona z menedżerem kolejek, pod warunkiem, że maksymalna liczba połączeń określonych przez właściwość maxConnection nie została przekroczona. W związku z tym maksymalna liczba komponentów MDB jest równa maksymalnej liczbie połączeń. Jeśli liczba wdrożonych komponentów MDB osiągnie tę wartość maksymalną, wszelkie próby wdrożenia innego komponentu MDB nie powiodą się. Jeśli komponent MDB jest zatrzymany, jego połączenie może być używane przez inny komponent MDB.

W ogólnym przypadku, jeśli wdrażana jest wiele komponentów MDB, należy zwiększyć wartość właściwości maxConnections .

Właściwości reconnectionRetryCount i reconnectionRetryokreślają zachowanie adaptera zasobów, gdy połączenia z menedżerem kolejek produktu IBM MQ nie powiodą się, ponieważ na przykład awaria sieci. Jeśli nawiązanie połączenia nie powiedzie się, adapter zasobów zawiesi dostarczanie komunikatów do wszystkich komponentów MDB dostarczonych przez to połączenie przez okres określony przez właściwość reconnectionRetry. Następnie adapter zasobów podejmuje próbę ponownego nawiązania połączenia z menedżerem kolejek. Jeśli próba nie powiedzie się, adapter zasobów podejmuje dalsze próby ponownego połączenia w odstępach czasu określonych przez właściwość reconnectionRetry, dopóki nie zostanie osiągnięty limit określony przez właściwość reconnectionRetryCount. Jeśli wszystkie próby nie powiodą się, dostarczenie zostanie zatrzymane na stałe do momentu ręcznego zrestartowania komponentów MDB.

W ogólnym przypadku obiekt ResourceAdapter nie wymaga administrowania. Aby na przykład włączyć śledzenie diagnostyczne w systemach UNIX and Linux , można ustawić następujące właściwości:

```
traceEnabled:    true
traceLevel:      10
```

Te właściwości nie mają wpływu na to, że adapter zasobów nie został uruchomiony, co ma miejsce na przykład wtedy, gdy aplikacje korzystające z zasobów produktu IBM MQ działają tylko w kontenerze klienta. W tej sytuacji można ustawić właściwości dla śledzenia diagnostycznego jako właściwości systemu Java Maszyna wirtualna (JVM). Właściwości można ustawić za pomocą opcji -D w komendzie **java** , jak w następującym przykładzie:

```
java ... -DtraceEnabled=true -DtraceLevel=6
```

Nie ma potrzeby definiowania wszystkich właściwości obiektu ResourceAdapter . Wszystkie właściwości pozostawione nieokreślonym przyjmują wartości domyślne. W środowisku zarządzanym lepiej nie mieszać dwóch sposobów określania właściwości. Jeśli zostaną one wymieszane, właściwości systemowe maszyny JVM mają pierwszeństwo przed właściwościami obiektu ResourceAdapter .

WebSphere Application Server traditional Konfiguracja

Te same właściwości są dostępne dla adaptera zasobów w produkcie WebSphere Application Server traditional, ale powinny być ustawione w panelu właściwości adaptera zasobów (patrz sekcja Ustawienia dostawcy JMS w dokumentacji produktu WebSphere Application Server traditional . Śledzenie jest sterowane przez sekcję diagnostyczną w konfiguracji produktu 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 za pomocą elementów XML w pliku `server.xml` , jak pokazano w poniższym przykładzie:

```
<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 jest włączone przez dodanie tego elementu 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.

Nowe właściwości ActivationSpec w produkcie JMS 2.0

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 .

Na przykład można założyć, że 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 63 na stronie 459.

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 64 na stronie 470, są używane. Więcej informacji na temat używania tych dwóch właściwości zawiera sekcja “Właściwości ActivationSpec connectionFactoryLookup i destinationLookup” na stronie 474.

Właściwości używane do tworzenia połączenia JMS z menedżerem kolejek produktu IBM MQ

Wszystkie właściwości w pliku Tabela 63 na stronie 459 są opcjonalne.

Tabela 63. 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** • Nazwa kolejki	Nazwa kolejki, z której konsument połączenia odbiera komunikaty subskrypcji trwałej
brokerCCSubbrokerCCSub [1]	Łańcuch	• **SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE** • Nazwa kolejki	Nazwa kolejki, z której konsument połączenia odbiera komunikaty nietrwałej subskrypcji
brokerControlbrokerControl [1]	Łańcuch	• **SYSTEM.BROKER.CONTROL.QUEUE** • Nazwa kolejki	Nazwa kolejki sterującej brokera
brokerQueuekolejki brokera [1]	Łańcuch	• **"" (pusty łańcuch)** • Nazwa menedżera kolejek	Nazwa menedżera kolejek, w którym działa broker
brokerSubbrokerSub [1]	Łańcuch	• **SYSTEM.JMS.ND.SUBSCRIBER.QUEUE** • Nazwa kolejki	Nazwa kolejki, z której konsument komunikatów nietrwałych odbiera komunikaty

Tabela 63. Właściwości obiektu ActivationSpec , które są używane do tworzenia połączenia JMS (kontynuacja)

Nazwa właściwości	Typ	Poprawne wartości (wartość domyślna pogrubiona)	Opis
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. • **V1** -aby użyć brokera IBM MQ publish/subscribe broker.This jest wartością domyślną, jeśli właściwość TRANSPORT ma wartość BIND lub CLIENT. • **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** • 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** • Identyfikator kodowanego zestawu znaków obsługiwany przez wirtualną maszynę języka Java (JVM)	Identyfikator kodowanego zestawu znaków dla połączenia
kanał	Łańcuch	• **SYSTEM.DEF.SVRCONN** • Nazwa kanału MQI	Nazwa kanału MQI, który ma być używany
cleanupInterval [1]	int	• **3 600 000** • Dodatnia liczba całkowita	Odstęp czasu (w milisekundach) między uruchomieniami w tle programu narzędziowego procedury czyszczącej publikowania/subskrypcji
cleanupLevel [1]	Łańcuch	• **Bezpieczne** • BRAK • silny • Wymuszenie • NIEDUR	Poziom procedury czyszczącej dla składnicy subskrypcji opartej na brokerze
clientID	Łańcuch	• **Null** • Identyfikator klienta	Identyfikator klienta dla połączenia

Tabela 63. Właściwości obiektu ActivationSpec , które są używane do tworzenia połączenia JMS (kontynuacja)

Nazwa właściwości	Typ	Poprawne wartości (wartość domyślna pogrubiona)	Opis
cloneSupport	Łańcuch	• **DISABLED** -w danym momencie może działać tylko jedna instancja trwałego subskrybenta tematów. • 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.
Wyszukiwanie connectionFactory	Łańcuch	• **Null** • 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 474.

Tabela 63. 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)** • Łańcuch składający się z elementów oddzielonych przecinkami, w którym każdy element ma następujący format: `HOSTNAME(PORT)` gdzie *NAZWAHOSTA* jest nazwą DNS lub adresem IP.	Lista nazw połączeń TCP/IP używanych dla komunikacji przychodzącej. Jeśli ta opcja jest określona, **connectionNameList** zastępuje właściwości **hostname** i **port** . Ta właściwość jest używana do ponownego nawiązywania połączenia z menedżerami kolejek z wieloma instancjami. 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	wartość boolowska	• **Prawda** • Fałsz	Określa, czy wywołania niektórych metod nie powiodą się, jeśli menedżer kolejek jest w stanie wyciszania
headerCompression	Łańcuch	• **Brak** • SYSTEM-kompresja nagłówka komunikatu RLE jest wykonywana	Lista technik, których można użyć do kompresji danych nagłówka w połączeniu
hostName	Łańcuch	• **localhost** • Nazwa hosta • Adres IP	Nazwa hosta lub adres IP systemu, w którym znajduje się menedżer kolejek. Właściwości **hostname** i **port** są zastępowane przez właściwość **connectionNameList** , gdy jest ona określona.

Tabela 63. Właściwości obiektu ActivationSpec , które są używane do tworzenia połączenia JMS (kontynuacja)

Nazwa właściwości	Typ	Poprawne wartości (wartość domyślna pogrubiona)	Opis
localAddress	Łańcuch	• **Null** • Łańcuch w formacie: `[ host_name ][(low_port [, high_port ])]` 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: • Lokalny interfejs sieciowy, który ma być używany • Port lokalny lub zakres portów lokalnych, które mają być używane 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** • Lista zawierająca jedną lub więcej następujących wartości rozdzielonych znakami odstępu: RLE ZLIBFAST ZLIBHIGH	Lista technik, których można użyć do kompresowania danych komunikatu w połączeniu
messageRetention [1]	wartość boolowska	• **true** (prawda)-w kolejce wejściowej pozostają niepotrzebne komunikaty. • false-Nieżądane komunikaty są przetwarzane zgodnie z ich opcjami dyspozycji	Określa, czy konsument połączenia przechowuje niepożądane komunikaty w kolejce wejściowej.
messageSelection [1]	Łańcuch	• **client** • 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.

Tabela 63. Właściwości obiektu ActivationSpec , które są używane do tworzenia połączenia JMS (kontynuacja)

Nazwa właściwości	Typ	Poprawne wartości (wartość domyślna pogrubiona)	Opis
Hasło	Łańcuch	• **Null** • Hasło	Domyślne hasło używane podczas tworzenia połączenia z menedżerem kolejek
pollingInterval [1]	int	• **5000** • 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** • Numer portu TCP	Port, na którym nasłuchuje menedżer kolejek. Właściwości **hostname** i **port** są zastępowane przez właściwość **connectionNameList** , gdy jest ona określona.
providerVersion	string (łańcuch)	• **nieokreślona** • Łańcuch w jednym z następujących formatów – V.R.M.F – V.R.M – V.R – V 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)** • Nazwa menedżera kolejek	Nazwa menedżera kolejek, z którym ma zostać nawiązane połączenie

Tabela 63. Właściwości obiektu ActivationSpec , które są używane do tworzenia połączenia JMS (kontynuacja)

Nazwa właściwości	Typ	Poprawne wartości (wartość domyślna pogrubiona)	Opis
receiveExit [3]	Łańcuch	• **Null** • 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** • 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 [1]	int	• **5000** • 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.

Tabela 63. Właściwości obiektu ActivationSpec , które są używane do tworzenia połączenia JMS (kontynuacja)

Nazwa właściwości	Typ	Poprawne wartości (wartość domyślna pogrubiona)	Opis
securityExit [3]	Łańcuch	• **Null** • Pełna nazwa klasy implementującej interfejs IBM MQ classes for Java , MQSecurityExit	Identyfikuje program obsługi wyjścia zabezpieczeń kanału
securityExit-inicjowanie	Łańcuch	• **Null** • Łańcuch danych użytkownika	Dane użytkownika przekazywane do programu obsługi wyjścia zabezpieczeń kanału podczas jego wywołania
sendExit [3]	Łańcuch	• **Null** • 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** • 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	wartość boolowska	• NIE-Połączenie klienta nie może współużytkować swojego gniazda. • **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]	wartość boolowska	• **false** -subskrypcje otrzymują często zgodne komunikaty. • true-subskrypcje otrzymują rzadko zgodne komunikaty. Ta wartość wymaga, aby kolejka subskrypcji mogła zostać otwarta do przeglądania.	Steruje strategią pobierania komunikatów obiektu TopicSubscriber

Tabela 63. Właściwości obiektu ActivationSpec , które są używane do tworzenia połączenia JMS (kontynuacja)

Nazwa właściwości	Typ	Poprawne wartości (wartość domyślna pogrubiona)	Opis
Magazyny sslCert	Łańcuch	• **Null** • Łańcuch zawierający jeden lub więcej adresów URL LDAP rozdzielonych odstępami. Każdy URL LDAP ma następujący format: `ldap://host_name [: port ]` 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** • Nazwa CipherSuite	Zestaw algorytmów szyfrowania CipherSuite , który ma być używany na potrzeby połączenia TLS
sslFipsWymagane [2]	wartość boolowska	• **Fałsz** • Prawda	Określa, czy połączenie TLS musi używać pakietu CipherSuite obsługiwanego przez dostawcę IBM Java JSSE FIPS (IBMJSSEFIPS).
SSLPEERNAME	Łańcuch	• **Null** • 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** • 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 .

Tabela 63. Właściwości obiektu ActivationSpec , które są używane do tworzenia połączenia JMS (kontynuacja)

Nazwa właściwości	Typ	Poprawne wartości (wartość domyślna pogrubiona)	Opis
statusRefreshPrzedział czasu [1]	int	• **60000** • 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** • MIGRATE • QUEUE	Określa miejsce, w którym program IBM MQ classes for JMS zapisuje dane trwałe dotyczące aktywnych subskrypcji.

Tabela 63. Właściwości obiektu ActivationSpec , które są używane do tworzenia połączenia JMS (kontynuacja)

Nazwa właściwości	Typ	Poprawne wartości (wartość domyślna pogrubiona)	Opis
transportType	Łańcuch	• **client** • POWIĄZANIA • 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. 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** • Nazwa użytkownika	Domyślna nazwa użytkownika, która ma być używana podczas tworzenia połączenia z menedżerem kolejek

Tabela 63. 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 • **TOPIC** -rozpoznaje tylko znaki wieloznaczne na poziomie tematu używane w wersji 2 brokera.	Która wersja składni znaku wieloznacznego ma być używana

Uwagi:

1. Ta właściwość może być używana z wersją 7.0 produktu IBM MQ classes for JMS. Nie ma to wpływu na aplikację połączoną z menedżerem kolejek produktu IBM WebSphere MQ 7.0 , chyba że dla właściwości **providerVersion** ustawiono numer wersji mniejszy niż 7.
2. Ważne informacje na temat używania wymaganej właściwości sslFipszawiera sekcja "Ograniczenia adaptera zasobów produktu IBM MQ" na stronie 446.
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 kanału" na stronie 277.

Właściwości używane do tworzenia konsumenta połączenia JMS

Uwaga: Wartości **destination** i **destinationType** muszą być zdefiniowane jawnie. Wszystkie pozostałe właściwości w pliku Tabela 64 na stronie 470 są opcjonalne.

Tabela 64. 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** • 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 474.
destinationType	Łańcuch	• javax.jms.Queue • javax.jms.Topic	Typ miejsca docelowego, kolejki lub tematu

Tabela 64. Właściwości obiektu ActivationSpec , które są używane do tworzenia konsumenta połączenia JMS (kontynuacja)

Nazwa właściwości	Typ	Poprawne wartości (wartość domyślna pogrubiona)	Opis
maxMessages	int	• **1** • 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** • Dodatnia liczba całkowita	Maksymalna liczba sesji serwera w puli sesji serwera używanych przez konsumenta połączenia
messageSelector	Łańcuch	• **Null** • Wyrażenie selektora komunikatów SQL92	Wyrażenie selektora komunikatów określające, które komunikaty mają zostać dostarczone
nonASFTimeout	int	• **0** • 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. Ten parametr jest poprawny, jeśli: • Aplikacja działa w systemie WebSphere Application Server 7.0 lub nowszym. • 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 448.
Włączona opcja nonASFRollback	wartość boolowska	• **false** (fałsz)-komunikat jest pobierany nawet wtedy, gdy komponent MDB zakończy działanie niepowodzeniem. • 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** • Dodatnia liczba całkowita	Czas (w milisekundach), przez który nieużywana sesja serwera jest utrzymywana w puli sesji serwera przed zamknięciem z powodu nieaktywności

Tabela 64. Właściwości obiektu ActivationSpec , które są używane do tworzenia konsumenta połączenia JMS (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. • DISABLED-odczyt z wyprzedzeniem jest niedozwolony. • ENABLED-odczyt z wyprzedzeniem jest dozwolony. • QUEUE-określ, czy odczyt z wyprzedzeniem jest dozwolony, odwołując się do definicji kolejki. • TOPIC-Określenie, czy odczyt z wyprzedzeniem jest dozwolony, poprzez odwołanie się do definicji tematu.	Określa, czy komponent MDB może używać odczytu z wyprzedzeniem do pobierania nietrwałych komunikatów z miejsca docelowego do buforu wewnętrznego przed ich odebraniem.
readAheadClosePolicy	int	• **ALL** -wszystkie komunikaty w wewnętrznym buforze odczytu z wyprzedzeniem są dostarczane do komponentu MDB przed jego zatrzymaniem. • 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.
receiveCCSID	int	• **0** -użyj maszyny JVM `Charset.defaultCharset` • 1208- UTF-8 • 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** • QMGR	Właściwość miejsca docelowego, która określa, czy konwersja danych będzie wykonywana przez menedżer kolejek.

Tabela 64. Właściwości obiektu ActivationSpec , które są używane do tworzenia konsumenta połączenia JMS (kontynuacja)

Nazwa właściwości	Typ	Poprawne wartości (wartość domyślna pogrubiona)	Opis
sharedSubscription	wartość boolowska	• **False** -komponent MDB nie powinien otwierać subskrypcji jako subskrypcji współużytkowanej. • 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 476.
startTimeout	int	• **10 000** • 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. • 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)** • Nazwa subskrypcji	Nazwa trwałej subskrypcji
useJNDI	wartość boolowska	• **false** -właściwość o nazwie destination jest interpretowana jako nazwa kolejki lub tematu produktu IBM MQ . • true-właściwość o nazwie destination jest interpretowana jako nazwa obiektu javax.jms.Queue lub obiektu javax.jms.Topic w przestrzeni nazw JNDI serwera aplikacji.	Określa sposób interpretowania wartości właściwości o nazwie destination. **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 definiowania 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

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 63 na stronie 459 i Tabela 64 na stronie 470.

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 63 na stronie 459. 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 został on ustawiony, obiekt docelowy o nazwie JNDI określonej przez właściwość jest wyszukiwany w pliku JNDI. Ta właściwość ma pierwszeństwo przed właściwością **useJNDI** .

  Właściwość useJNDI jest nieaktualna w wersji IBM MQ 9.0 , ponieważ właściwość **destinationLookup** jest odpowiednikiem specyfikacji JMS 2.0 wykonywania tej samej funkcji.

Właściwości ActivationSpec bez odpowiedników w produkcie IBM MQ classes for JMS

Większość właściwości obiektu ActivationSpec jest równoważna właściwościom obiektów lub parametrów metod IBM MQ classes for JMS systemu IBM MQ classes for JMS . Jednak trzy właściwości strojenia i jedna właściwość łatwości używania nie mają odpowiedników w produkcie IBM MQ classes for JMS:

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 używana 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 64 na stronie 470.

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:

```
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 rozproszonych transakcji, ale adapter zasobów nie obsługuje rozproszonych transakcji. Informacje na temat instalowania adaptera zasobów w celu obsługi transakcji rozproszonych zawiera sekcja “Instalowanie adaptera zasobów produktu IBM MQ” na stronie 450.

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 223.

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

FIPS (Federal Information Processing Standards) dla systemów UNIX, Linux i Windows

Przykłady definiowania właściwości sharedSubscription

Właściwość sharedSubscription specyfikacji aktywowania można zdefiniować w pliku WebSphere Liberty `server.xml` . Alternatywnie można zdefiniować właściwość w komponencie bean sterowanym komunikatami (MDB) za pomocą adnotacji.

Przykład: definiowanie w pliku Liberty `server.xml`

W pliku WebSphere Liberty `server.xml` definiuje się specyfikację aktywowania, jak pokazano w poniższym przykładzie. W tym przykładzie tworzona jest trwała subskrypcja współużytkowana do menedżera kolejek na hoście localhost/port 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 komponencie MDB

Istnieje również możliwość zdefiniowania właściwości sharedSubscription w komponencie MDB za pomocą adnotacji, jak pokazano w poniższym przykładzie:

```
@ActioncationConfigProperty(propertyName ="sharedSubscription",
propertyValue = "true")
```

W poniższym przykładzie przedstawiono fragment kodu komponentu MDB, który korzysta z metody adnotacji:

```
/**
 * 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

Subskrybenty i subskrypcje

Trwałość subskrypcji

“Klonowane i współużytkowane subskrypcje” na stronie 323

W produkcie IBM MQ 8.0 lub nowszym istnieją dwie metody dające wielu konsumentom dostęp do tej samej subskrypcji. Te dwie metody są używane przy użyciu sklonowanych subskrypcji lub przy użyciu 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:

```
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, a następnie 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 65 na stronie 478 zawiera listę właściwości obiektu ConnectionFactory . Serwer aplikacji używa tych właściwości do utworzenia obiektu JMS ConnectionFactory .

Tabela 65. 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**).
brokerCCSubbrokerCCSub [1]	Łańcuch	• **SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE** • Nazwa kolejki	Nazwa kolejki, z której konsument połączenia odbiera komunikaty nietrwałej subskrypcji.
brokerControlbrokerControl [1]	Łańcuch	• **SYSTEM.BROKER.CONTROL.QUEUE** • Nazwa kolejki	Nazwa kolejki sterującej brokera.

Tabela 65. Właściwości obiektu ConnectionFactory (kontynuacja)

Nazwa właściwości	Typ	Poprawne wartości (wartość domyślna pogrubiona)	Opis
brokerPubKolejka [1]	Łańcuch	• **SYSTEM.BROKER.DEFAULT.STREAM** • Nazwa kolejki	Nazwa kolejki, do której wysyłane są opublikowane komunikaty (kolejka strumienia).
brokerQueuekolejki brokera [1]	Łańcuch	• **"" (pusty łańcuch)** • Nazwa menedżera kolejek	Nazwa menedżera kolejek, w którym działa broker.
brokerSubbrokerSub [1]	Łańcuch	• **SYSTEM.JMS.ND.SUBSCRIBER.QUEUE** • Nazwa kolejki	Nazwa kolejki, z której konsument nietrwałych komunikatów odbiera komunikaty. Więcej informacji na ten temat zawiera opis właściwości BROKERSUBQ .
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. • **V1** -aby użyć brokera publikowania/ subskrypcji produktu IBM MQ . Ta wartość jest wartością domyślną, jeśli dla parametru TRANSPORT ustawiono wartość BIND lub CLIENT. • **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** • Adres URL (Uniform Resource Locator)	Adres URL identyfikujący nazwę i położenie pliku zawierającego tabelę definicji kanału klienta (CCDT) oraz określający sposób dostępu do pliku.
CCSID	Łańcuch	• **819** • Identyfikator kodowanego zestawu znaków obsługiwany przez wirtualną maszynę języka Java (JVM)	Identyfikator kodowanego zestawu znaków dla połączenia.
kanał	Łańcuch	• **SYSTEM.DEF.SVRCONN** • Nazwa kanału MQI	Nazwa kanału MQI, który ma być używany.
cleanupInterval [1]	int	• **3 600 000** • Dodatnia liczba całkowita	Odstęp czasu (w milisekundach) między uruchomieniami w tle programu narzędziowego do czyszczenia publikowania/ subskrypcji.

Tabela 65. Właściwości obiektu ConnectionFactory (kontynuacja)

Nazwa właściwości	Typ	Poprawne wartości (wartość domyślna pogrubiona)	Opis
cleanupLevel [1]	Łańcuch	• **Bezpieczne** • BRAK • silny • Wymuszenie • NIEDUR	Poziom procedury czyszczącej dla składnicy subskrypcji opartej na brokerze.
clientID	Łańcuch	• **Null** • Identyfikator klienta	Identyfikator klienta dla połączenia.
cloneSupport	Łańcuch	• **DISABLED** -w danym momencie może działać tylko jedna instancja trwałego subskrybenta tematów. • 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)** • Łańcuch składający się z elementów oddzielonych przecinkami, w którym każdy element ma następujący format: `HOSTNAME(PORT)` gdzie *NAZWAHOSTA* jest nazwą DNS lub adresem IP.	Lista nazw połączeń TCP/IP używanych do komunikacji wychodzącej. **connectionNameList** zastępuje właściwości **hostname** i **port** . Ta właściwość jest używana do ponownego nawiązywania połączenia z menedżerami kolejek z wieloma instancjami. 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	wartość boolowska	• **Prawda** • Fałsz	Określa, czy wywołania niektórych metod nie powiodą się, jeśli menedżer kolejek jest w stanie wyciszania.
headerCompression	Łańcuch	• **Brak** • Kompresja nagłówka komunikatu SYSTEM-RLE jest wykonywana.	Lista technik, których można użyć do kompresji danych nagłówka w połączeniu.

Tabela 65. Właściwości obiektu ConnectionFactory (kontynuacja)

Nazwa właściwości	Typ	Poprawne wartości (wartość domyślna pogrubiona)	Opis
hostName	Łańcuch	• **localhost** • Nazwa hosta • Adres IP	Nazwa hosta lub adres IP systemu, w którym znajduje się menedżer kolejek. Właściwości **hostname** i **port** są zastępowane przez właściwość **connectionNameList** , gdy jest ona określona.
localAddress	Łańcuch	• **Null** • Łańcuch w formacie: `[ host_name ][(low_port [, high_port ])]` 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: • Lokalny interfejs sieciowy, który ma być używany • Port lokalny lub zakres portów lokalnych, które mają być używane 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** • Lista zawierająca jedną lub więcej następujących wartości rozdzielonych znakami odstępu: RLE ZLIBFAST ZLIBHIGH	Lista technik, których można użyć do kompresji danych komunikatu w połączeniu.
messageSelection [1]	Łańcuch	• **client** • 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 **brokerVersion** ma wartość 1.
Hasło	Łańcuch	• **Null** • Hasło	Hasło domyślne używane podczas tworzenia połączenia z menedżerem kolejek.

Tabela 65. Właściwości obiektu ConnectionFactory (kontynuacja)

Nazwa właściwości	Typ	Poprawne wartości (wartość domyślna pogrubiona)	Opis
pollingInterval [1]	int	• **5000** • 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** • Numer portu TCP	Port, na którym nasłuchuje menedżer kolejek. Właściwości **hostname** i **port** są zastępowane przez właściwość **connectionNameList** , gdy jest ona określona.
providerVersion	string (łańcuch)	• **nieokreślona** • Łańcuch w jednym z następujących formatów – V.R.M.F – V.R.M – V.R – V 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.
pubAckprzedział_czasu [1]	int	• **25** • Dodatnia liczba całkowita	Liczba komunikatów opublikowanych przez publikator, zanim produkt IBM MQ classes for JMS zażąda potwierdzenia od brokera.
queueManager	Łańcuch	• **"" (pusty łańcuch)** • Nazwa menedżera kolejek	Nazwa menedżera kolejek, z którym ma zostać nawiązane połączenie.

Tabela 65. Właściwości obiektu ConnectionFactory (kontynuacja)			
Nazwa właściwości	**Typ**	**Poprawne wartości (wartość domyślna pogrubiona)**	**Opis**
receiveExit [3]	Łańcuch	• **Null** • 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** • 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 [1]	int	• **5000** • 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** • Pełna nazwa klasy implementującej interfejs IBM MQ classes for Java , MQSecurityExit	Identyfikuje program obsługi wyjścia zabezpieczeń kanału.
securityExit-inicjowanie	Łańcuch	• **Null** • Łańcuch danych użytkownika	Dane użytkownika przekazywane do programu obsługi wyjścia zabezpieczeń kanału podczas jego wywołania.

Tabela 65. Właściwości obiektu ConnectionFactory (kontynuacja)

Nazwa właściwości	Typ	Poprawne wartości (wartość domyślna pogrubiona)	Opis
SENDCHECKCOUNT	int	• **0** • 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** • 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** • 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	wartość boolowska	• NIE-Połączenie klienta nie może współużytkować swojego gniazda. • **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]	wartość boolowska	• **false** -subskrypcje otrzymują często zgodne komunikaty. • 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** • Łańcuch zawierający jeden lub więcej adresów URL LDAP rozdzielonych odstępami. Każdy adres URL LDAP ma następujący format: `ldap://host_name [: port ]` 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** • Nazwa CipherSuite	CipherSuite , który ma być używany dla połączenia TLS.

Tabela 65. Właściwości obiektu ConnectionFactory (kontynuacja)

Nazwa właściwości	Typ	Poprawne wartości (wartość domyślna pogrubiona)	Opis
sslFipsWymagane [2]	wartość boolowska	• **Fałsz** • Prawda	Określa, czy połączenie TLS musi używać pakietu CipherSuite obsługiwanego przez dostawcę IBM Java JSSE FIPS (IBMJSSEFIPS).
SSLPEERNAME	Łańcuch	• **Null** • 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** • 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 .
statusRefreshPrzedział czasu [1]	int	• **60000** • 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 [1]	Łańcuch	• **BROKER** • MIGRATE • QUEUE	Określa miejsce, w którym program IBM MQ classes for JMS zapisuje dane trwałe dotyczące aktywnych subskrypcji.

Tabela 65. Właściwości obiektu ConnectionFactory (kontynuacja)

Nazwa właściwości	Typ	Poprawne wartości (wartość domyślna pogrubiona)	Opis
targetClientZgodne	wartość boolowska	• **Prawda** • 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 . Tę właściwość można również skonfigurować dla specyfikacji aktywowania. Więcej informacji na ten temat zawiera “Konfigurowanie właściwości targetClientMatching dla specyfikacji aktywowania” na stronie 496.

Tabela 65. Właściwości obiektu ConnectionFactory (kontynuacja)

Nazwa właściwości	Typ	Poprawne wartości (wartość domyślna pogrubiona)	Opis
temporaryModel	Łańcuch	• **SYSTEM.DEFAULT.MODEL.QUEUE** • SYSTEM.JMS.TEMPQ.MODEL • Dowolny łańcuch	Nazwa kolejki modelowej, na podstawie której są tworzone tymczasowe kolejki produktu JMS . Użyj SYSTEM.DEFAULT.MODEL.QUEUE , jeśli spełnione są oba poniższe warunki: • Aplikacja używa kolejki tymczasowej, która będzie akceptować komunikaty nietrwałe. • 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. Użyj SYSTEM.JMS.TEMPQ.MODEL w następujących sytuacjach: • Gdy aplikacja używa kolejki tymczasowej, która będzie akceptować komunikaty trwałe. • Jeśli wiele aplikacji może nawiązać połączenie z menedżerem kolejek, który wskazuje fabryka połączeń ConnectionFactory , a te aplikacje muszą jednocześnie tworzyć kolejki tymczasowe. Zdefiniuj nową kolejkę modelową z atrybutem **DEFPSIST** ustawionym na YESi atrybutem **DEFSOPT** ustawionym na SHARED w następującej sytuacji: • 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. Podczas tworzenia nowej kolejki modelowej należy ustawić właściwość **temporaryModel** na nazwę nowej kolejki modelowej.

Tabela 65. Właściwości obiektu ConnectionFactory (kontynuacja)

Nazwa właściwości	Typ	Poprawne wartości (wartość domyślna pogrubiona)	Opis
tempQPrefix	Łańcuch	• **"" (pusty łańcuch)** • 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 65. Właściwości obiektu ConnectionFactory (kontynuacja)			
Nazwa właściwości	**Typ**	**Poprawne wartości (wartość domyślna pogrubiona)**	**Opis**
transportType	Łańcuch	• **client** • POWIĄZANIA • 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. 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** • 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 • TOPIC-rozpoznaje tylko znaki wieloznaczne na poziomie tematu, używane w brokerze w wersji 2	Która wersja składni znaku wieloznacznego ma być używana.

Uwagi:

1. Ta właściwość może być używana z wersją 7.0 produktu IBM WebSphere MQ classes for JMS , ale nie ma wpływu na aplikację połączoną z menedżerem kolejek produktu IBM WebSphere MQ 7.0 , chyba że dla właściwości providerVersion ustawiono numer wersji mniejszy niż 7.
2. Ważne informacje na temat używania wymaganej właściwości sslFipszawiera sekcja “Ograniczenia adaptera zasobów produktu IBM MQ” na stronie 446.

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 kanału” na stronie 277.

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 66 na stronie 490 zawiera listę właściwości wspólnych dla obiektu kolejki i obiektu tematu.

Tabela 66. Właściwości wspólne dla obiektu kolejki i obiektu tematu

Nazwa właściwości	Typ	Poprawne wartości (wartość domyślna pogrubiona)	Opis
CCSID	Łańcuch	• **1208** • Identyfikator kodowanego zestawu znaków obsługiwany przez wirtualną maszynę języka Java (JVM)	Identyfikator kodowanego zestawu znaków dla miejsca docelowego.
kodowanie	Łańcuch	• **native** • Łańcuch składający się z trzech znaków: – Pierwszy znak określa reprezentację binarnych liczb całkowitych: - *N* oznacza normalne kodowanie. - *R* oznacza kodowanie odwrotne. – Drugi znak określa reprezentację upakowanych dziesiętnych liczb całkowitych: - *N* oznacza normalne kodowanie. - *R* oznacza kodowanie odwrotne. – Trzeci znak określa reprezentację liczb zmiennopozycyjnych: - *N* oznacza standardowe kodowanie IEEE. - *R* oznacza odwrotne kodowanie IEEE. - *3* oznacza kodowanie zSeries . NATIVE jest odpowiednikiem łańcucha NNN.	Reprezentacja binarnych liczb całkowitych, upakowanych liczb dziesiętnych i liczb zmiennopozycyjnych dla miejsca docelowego.

Tabela 66. Właściwości wspólne dla obiektu kolejki i obiektu tematu (kontynuacja)

Nazwa właściwości	Typ	Poprawne wartości (wartość domyślna pogrubiona)	Opis
Utrata ważności	Łańcuch	• **APP** -czas utraty ważności komunikatu jest określany przez producenta komunikatów. • UNLIM-Komunikat nigdy nie traci ważności. • 0-komunikat nigdy nie traci ważności. • Dodatnia liczba całkowita reprezentująca czas utraty ważności komunikatu w milisekundach.	Czas utraty ważności komunikatu wysłanego do miejsca docelowego.
FAILIFQUIESCE	Łańcuch	• **Prawda** • Fałsz	Określa, czy próba uzyskania dostępu do miejsca docelowego nie powiedzie się, jeśli menedżer kolejek jest w stanie wyciszania.

Tabela 66. Właściwości wspólne dla obiektu kolejki i obiektu tematu (kontynuacja)

Nazwa właściwości	Typ	Poprawne wartości (wartość domyślna pogrubiona)	Opis
Styl messageBody	Łańcuch	• **Nieokreślone** • JMS • MQ	W kolejkach i tematach w systemie JMS można ustawić właściwość **`messageBodyStyle`** : UNSPECIFIED (wartość domyślna) • Podczas wysyłania produkt IBM MQ classes for JMS generuje i dołącza nagłówek MQRFH2 w zależności od wartości właściwości WMQ_TARGET_CLIENT. • 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 . JMS • Podczas wysyłania produkt IBM MQ classes for JMS automatycznie generuje nagłówek MQRFH2 i dołącza nagłówek do komunikatu IBM MQ . • 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 . MQ • Podczas wysyłania IBM MQ classes for JMS nie generuj nagłówka MQRFH2. • Podczas odbierania produkt IBM MQ classes for JMS przedstawia MQRFH2 jako część treści komunikatu JMS .

Tabela 66. 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. • QDEF-trwałość komunikatu jest określana przez atrybut **DefPersistence** kolejki IBM MQ . • PERS-komunikat jest trwały. • NON-Komunikat jest nietrwały. • HIGH-Trwałość komunikatu jest określana przez atrybut **NonPersistentMessageClass** kolejki IBM MQ zgodnie z wyjaśnieniem w sekcji "Komunikaty trwałe produktu JMS" na stronie 241.	Trwałość komunikatu wysłanego do miejsca docelowego.
priorytet	Łańcuch	• **APP** -priorytet komunikatu jest określany przez producenta komunikatu. • QDEF-priorytet komunikatu jest określany przez atrybut **DefPriority** kolejki IBM MQ . • 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. • TOPIC-Określenie, czy asynchroniczne operacje umieszczania są dozwolone, poprzez odwołanie się do definicji tematu. • DESTINATION-określa, czy asynchroniczne operacje umieszczania są dozwolone, odwołując się do definicji kolejki lub tematu. • DISABLED-asynchroniczne operacje umieszczania nie są dozwolone. • ENABLED-asynchroniczne operacje put są dozwolone.	Określa, czy producenci komunikatów mogą używać asynchronicznych operacji umieszczania w celu wysyłania komunikatów do tego miejsca docelowego.

Tabela 66. 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. • DISABLED-odczyt z wyprzedzeniem jest niedozwolony. • ENABLED-odczyt z wyprzedzeniem jest dozwolony. • QUEUE-określ, czy odczyt z wyprzedzeniem jest dozwolony, odwołując się do definicji kolejki. • 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` • 1208- UTF-8 • 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** • QMGR	Właściwość miejsca docelowego, która określa, czy konwersja danych będzie wykonywana przez menedżer kolejek.
targetClient	Łańcuch	• **JMS** -celem komunikatu jest aplikacja JMS . • 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 67 na stronie 494 zawiera listę właściwości, które są specyficzne dla obiektu kolejki.

Tabela 67. Właściwości specyficzne dla obiektu kolejki

Nazwa właściwości	Typ	Poprawne wartości (wartość domyślna pogrubiona)	Opis
baseQueueManagerName	Łańcuch	• **"" (pusty łańcuch)** • Nazwa menedżera kolejek	Nazwa menedżera kolejek, który jest właścicielem bazowej kolejki produktu IBM MQ .
Nazwa baseQueue	Łańcuch	• **"" (pusty łańcuch)** • Nazwa kolejki	Nazwa bazowej kolejki produktu IBM MQ .

Tabela 68 na stronie 495 zawiera listę właściwości, które są specyficzne dla obiektu Topic.

Tabela 68. Właściwości specyficzne dla obiektu Topic

Nazwa właściwości	Typ	Poprawne wartości (wartość domyślna pogrubiona)	Opis
baseTopicNazwa	Łańcuch	• **"" (pusty łańcuch)** • Nazwa tematu	Nazwa tematu bazowego.
brokerCCDurSubQueue [1]	Łańcuch	• **SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE** • Nazwa kolejki	Nazwa kolejki, z której konsument połączenia odbiera komunikaty subskrypcji trwałej.
brokerDurSubQueue [1]	Łańcuch	• **SYSTEM.JMS.D.SUBSCRIBER.QUEUE** • Nazwa kolejki	Nazwa kolejki, z której trwały subskrybent tematu odbiera komunikaty. Więcej informacji na ten temat zawiera opis właściwości **BROKEDURRSUBQ** w dokumentacji produktu IBM MQ Explorer .
brokerPubKolejka [1]	Łańcuch	• **Nie ustawiono** • 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, zostanie użyta wartość właściwości **brokerPubQueue** obiektu ConnectionFactory .
brokerPubQueueManager [1]	Łańcuch	• **"" (pusty łańcuch)** • Nazwa menedżera kolejek	Nazwa menedżera kolejek będącego właścicielem kolejki, do której są wysyłane komunikaty opublikowane w temacie.
brokerVersion [1]	Łańcuch	• **Nie ustawiono** • 1 • 2	Wersja używanego brokera. Wartość tej właściwości nadpisuje wartość właściwości **brokerVersion** obiektu ConnectionFactory . Jeśli jednak wartość tej właściwości nie zostanie ustawiona, zostanie użyta wartość właściwości **brokerVersion** obiektu ConnectionFactory .

Uwaga:

1. Ta właściwość może być używana z wersją 7.0 produktu IBM WebSphere MQ classes for JMS , ale nie ma wpływu na aplikację połączoną z menedżerem kolejek produktu IBM WebSphere MQ 7.0 , chyba że właściwość providerVersion obiektu ConnectionFactory ma numer wersji mniejszy niż 7.

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 UNIX, Linux, and Windows

V 9.1.0 *Konfigurowanie właściwości targetClientMatching dla specyfikacji aktywowania*

Istnieje możliwość skonfigurowania właściwości **targetClientMatching** dla specyfikacji aktywowania w taki sposób, aby nagłówek MQRFH2 był dołączany do komunikatów odpowiedzi, gdy komunikaty żądania nie zawierają nagłówka MQRFH2 . Oznacza to, że wszystkie właściwości komunikatu, które aplikacja definiuje w komunikacie odpowiedzi, są uwzględniane podczas wysyłania komunikatu.

O tym zadaniu

Jeśli aplikacja komponentu bean sterowanego komunikatami zużywa komunikaty, które nie zawierają nagłówka MQRFH2 , za pośrednictwem specyfikacji aktywowania adaptera zasobów JCA produktu IBM MQ , a następnie wysyła komunikaty odpowiedzi do miejsca docelowego JMS utworzonego w polu `JMSReplyTo` komunikatu żądania, komunikaty odpowiedzi muszą zawierać nagłówek MQRFH2 , nawet jeśli komunikaty żądania nie zawierają żadnych właściwości komunikatu, które aplikacja zdefiniowała w komunikacie odpowiedzi, są tracone.

Właściwość **targetClientMatching** definiuje, 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 wartość właściwości **targetClientMatching** zostanie ustawiona na `false`, nagłówek MQRFH2 może zostać umieszczony w komunikacie odpowiedzi wysłanym do miejsca docelowego JMS utworzonego z nagłówka JMSReplyTo komunikatu przychodzącego żądania, który nie zawiera MQRFH2. Dzieje się tak dlatego, 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 w komunikacie, gdy jest on wysyłany do kolejki produktu IBM MQ .

Jeśli właściwość **targetClientMatching** jest ustawiona na wartość `true` , a komunikat żądania nie zawiera nagłówka MQRFH2 , to nagłówek MQRFH2 nie jest zawarty w komunikacie odpowiedzi.

Procedura

- W produkcie WebSphere Application Server traditionalnależy użyć konsoli administracyjnej w celu zdefiniowania właściwości **targetClientMatching** jako właściwości niestandardowej w specyfikacji aktywowania produktu IBM MQ :

a) W panelu nawigacyjnym kliknij opcję **Resources-> JMS-> Activation specifications**(Zasoby-> JMS-> Specyfikacje aktywowania).

b) Wybierz nazwę specyfikacji aktywowania, która ma być wyświetlana lub zmieniona.

c) Kliknij opcję **Właściwości niestandardowe-> Nowy** , a następnie wprowadź szczegóły nowej właściwości niestandardowej.

Ustaw nazwę właściwości na `targetClientMatching`, typ `java.lang.Boolean` i wartość na `false`.

- W produkcie WebSphere Libertynależy określić właściwość **`targetClientMatching`** w definicji specyfikacji aktywowania w ramach produktu `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 212
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 477
Aby skonfigurować komunikację wychodzącą, należy zdefiniować właściwości obiektu ConnectionFactory i administrowanego obiektu docelowego.

V 9.1.1 *IBM MQ pauza komponentu bean sterowanego komunikatami w produkcie WebSphere Liberty*

Właściwość **`maxSequentialDeliveryFailures`** specyfikacji aktywowania definiuje maksymalną liczbę niepowodzeń dostarczania komunikatów sekwencyjnych do instancji komponentu bean sterowanego komunikatami (message-driven bean-MDB), która jest tolerowana przez adapter zasobów przed wstrzymaniem wstrzymania komponentu bean sterowanego komunikatami.

Zanim rozpoczniesz

Należy pamiętać o zestawie zdarzeń, które mogą spowodować wstrzymanie komponentu MDB w produkcie WebSphere Liberty. Adapter zasobów jest zdania, że jedna z następujących sytuacji jako niepowodzenie dostarczenia komunikatu:

- Z metody **`onMessage`** komponentu MDB został zgłoszony niekontrolowany wyjątek.
- `JMSException` występujący w trakcie przetwarzania adaptera zasobów przed dostarczeniem komunikatu do komponentu MDB.
- `JMSException` występujący w przetwarzaniu adaptera zasobów, po dostarczaniu komunikatu do komponentu MDB.
- Transakcja XA lub transakcja lokalna używana do wykorzystania wycofanego komunikatu.
- Brak dostępnego wątku na serwerze aplikacji w celu dostarczenia komunikatu do komponentu MDB.

O tym zadaniu

Wartością domyślną właściwości **`maxSequentialDeliveryFailures`** jest *-1*, co oznacza, że komponent MDB nigdy nie jest wstrzymany. Każda inna wartość ujemna jest traktowana tak samo jak *-1*. Wartość:

- *0* oznacza, że komponent MDB wstrzymuje pierwsze wystąpienie błędu.

- *1* oznacza, że komponent MDB jest wstrzymany na dwóch kolejnych błędach.
- *2* oznacza, że komponent MDB jest wstrzymany na trzech kolejnych błędach, itd.

Tę właściwość można skonfigurować dla specyfikacji aktywowania, tylko w produkcie WebSphere Liberty, a poziom Liberty ma wartość 18.0.0.4lub wyższą.

Ostrzeżenie: Jeśli atrybut zostanie ustawiony na wartość inną niż domyślna w dowolnym środowisku serwera aplikacji innym niż Liberty, wartość zostanie zignorowana, a do dziennika zostanie zapisany komunikat ostrzegawczy.

Ponadto możliwe jest zainstalowanie adaptera zasobów produktu IBM MQ w produkcie WebSphere Liberty jako ogólnego adaptera zasobów. Spowoduje to wyłączenie wszystkich możliwości integracji produktów IBM MQ i WebSphere Application Server oraz uniemożliwi możliwość wykrycia przez adapter zasobów, że jest on uruchomiony w produkcie Liberty. Oznacza to, że ustawienie parametru **maxSequentialDeliveryFailures** na wartość większą lub równą *0* nie jest obsługiwane i powoduje wygenerowanie komunikatu ostrzegawczego w dzienniku.

Procedura

- W produkcie WebSphere Libertynależy określić właściwość **maxSequentialDeliveryFailures** w definicji specyfikacji aktywowania w ramach produktu `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 477
Aby skonfigurować komunikację wychodzącą, należy zdefiniować właściwości obiektu ConnectionFactory i administrowanego obiektu docelowego.

Weryfikowanie instalacji adaptera zasobów

Program do sprawdzania poprawności instalacji (IVT) dla adaptera zasobów produktu IBM MQ jest dostarczany jako plik EAR. Aby korzystać z programu, należy go wdrożyć i zdefiniować niektóre obiekty jako zasoby produktu JCA .

O tym zadaniu

Program do sprawdzania poprawności instalacji (IVT) jest dostarczany jako plik archiwum korporacyjnego (EAR) o nazwie `wmq.jmsra.ivt.ear`. Ten plik jest instalowany z produktem IBM MQ classes for JMS w tym samym katalogu, w którym znajduje się plik RAR adaptera zasobów produktu IBM MQ `wmq.jmsra.rar`. Informacje o tym, gdzie te pliki są zainstalowane, zawiera sekcja “Instalowanie adaptera zasobów produktu IBM MQ” na stronie 450.

Należy wdrożyć program IVT na serwerze aplikacji. Program IVT zawiera serwlet oraz komponent MDB, który testuje, że komunikat może zostać wysłany do kolejki IBM MQ i odebrany z niej. Za pomocą programu IVT można sprawdzić, czy adapter zasobów IBM MQ został poprawnie skonfigurowany do obsługi transakcji rozproszonych. Jeśli adapter zasobów produktu IBM MQ jest wdrażany na serwerze aplikacji innej niżIBM , usługa IBM może poprosić użytkownika o przedstawienie narzędzia IVT w celu sprawdzenia, czy serwer aplikacji jest poprawnie skonfigurowany.

Przed uruchomieniem programu IVT należy zdefiniować obiekt ConnectionFactory , obiekt kolejki i ewentualnie obiekt specyfikacji aktywowania jako zasoby produktu JCA , a także upewnić się, że serwer aplikacji tworzy obiekty JMS z tych definicji i wiąże je z przestrzenią nazw produktu JNDI . Użytkownik

może wybrać właściwości obiektów, aby były zgodne z ustawieniami hosta i portu własnego obiektu QueueManager, ale następujący zestaw właściwości jest prostym przykładem:

```
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 specyfikacji ConnectionFactory, Queue and Activation Specification różni się w zależności od serwera aplikacji. Na przykład, aby ustawić te właściwości w produkcie WebSphere Liberty, należy dodać następujące wpisy do pliku `server.xml` serwera aplikacji:

```
<!-- 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, że obiekt ConnectionFactory będzie powiązany w przestrzeni nazw JNDI z nazwą jms/ivt/IVTCF i obiektem kolejki, który ma być powiązany z nazwą jms/ivt/IVTQueue. Można użyć różnych nazw, ale jeśli to zrobisz, musisz wprowadzić nazwy obiektów na początkowej stronie programu IVT i odpowiednio zmodyfikować plik EAR.

Po wdrożeniu programu IVT, gdy serwer aplikacji utworzył obiekty JMS i powiązano je z przestrzenią nazw JNDI , można uruchomić program IVT, wykonując następujące kroki.

Procedura

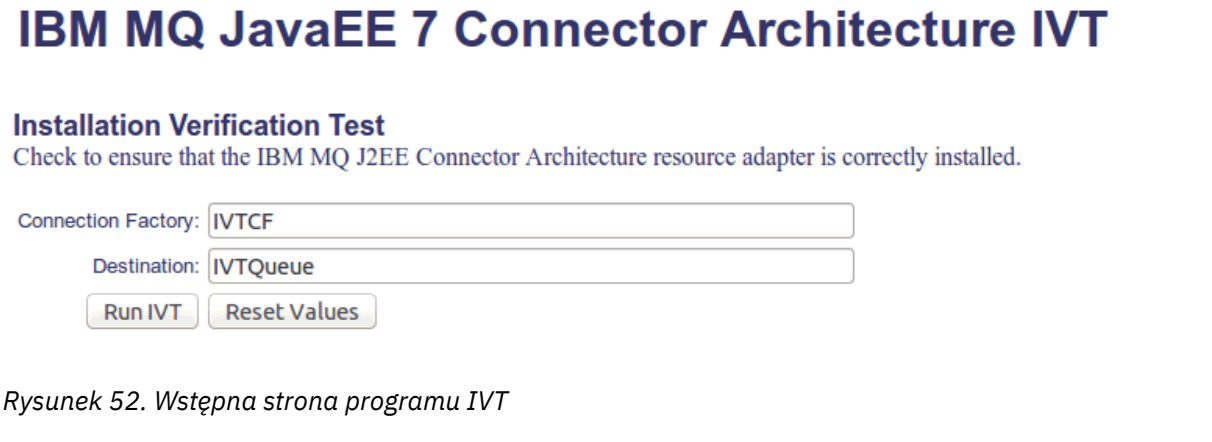
1. Uruchom program IVT, wprowadzając adres URL w następującym formacie w przeglądarce WWW:

   ```
   http://app_server_host: port/WMQ_IVT/
   ```

   gdzie *host_serwera_aplikacji* to adres IP lub nazwa 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/
   ```

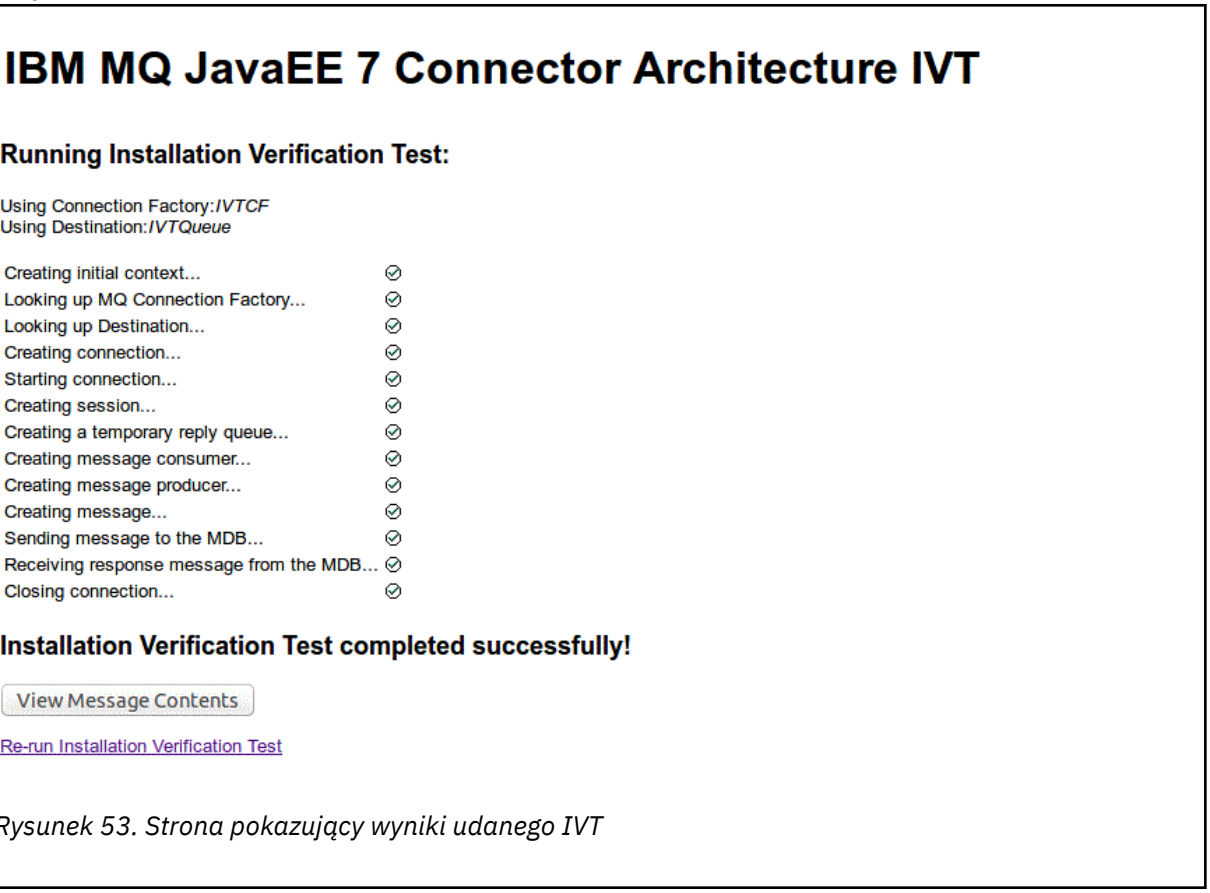
   W programie Rysunek 52 na stronie 500 wyświetlana jest początkowa strona programu IVT.



Rysunek 52. Wstępna strona programu IVT

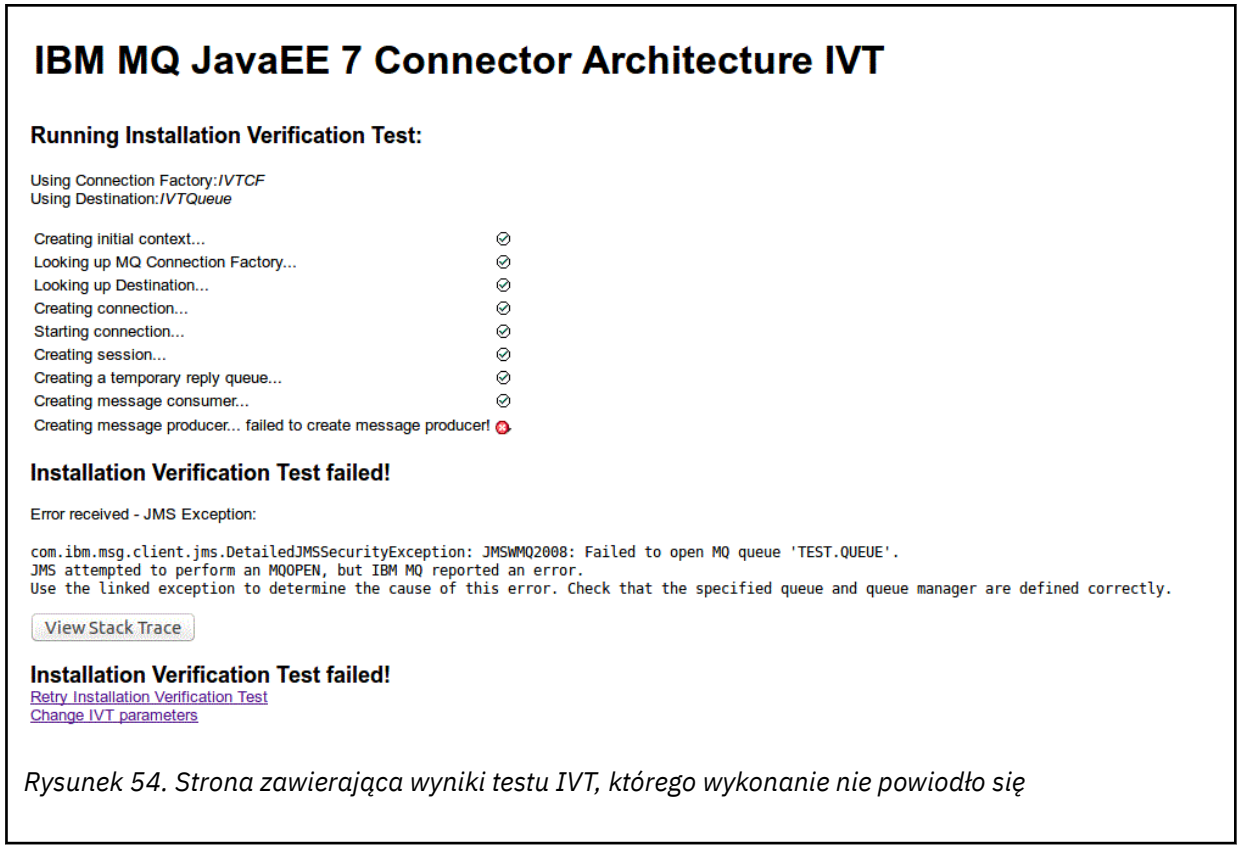
2. Aby uruchomić test, kliknij opcję **Uruchom program IVT**.

   Rysunek 53 na stronie 500 wyświetla stronę, która jest wyświetlana, jeśli program IVT zakończy się pomyślnie.



Rysunek 53. Strona pokazujący wyniki udanego IVT

Jeśli test IVT nie powiedzie się, zostanie wyświetlona strona taka jak wyświetlana w programie Rysunek 54 na stronie 501 . Aby uzyskać dalsze informacje na temat przyczyny niepowodzenia, kliknij opcję **Wyświetl stos wywołań**.



Rysunek 54. Strona zawierająca wyniki testu IVT, którego wykonanie nie powiodło się

Windows Instalowanie i testowanie adaptera zasobów na serwerze GlassFish Server

Aby zainstalować adapter zasobów IBM MQ na serwerze GlassFish Server w systemie operacyjnym Windows , należy najpierw utworzyć i uruchomić domenę. Następnie można wdrożyć i skonfigurować adapter zasobów, a następnie wdrożyć i uruchomić aplikację testową weryfikacyjną instalacji (IVT).

O tym zadaniu

Ważne: Te instrukcje są przeznaczone dla serwera GlassFish Server w wersji 4.

W przypadku tego zadania założono, że użytkownik korzysta z działającego serwera aplikacji serwera GlassFish Server, a użytkownik jest zaznajomiony z zadaniami administrowania standardowymi. W tym zadaniu założono również, że w systemie lokalnym jest dostępna instalacja produktu IBM MQ , a użytkownik jest zaznajomiony ze standardowymi zadaniami administracyjnymi.

Uwaga: Aby wykonać następujące kroki czynności, należy mieć działającą instalację produktu IBM MQ z następującymi skonfigurowanymi obiektami:

- Menedżer kolejek o nazwie QM, który jest uruchamiany na porcie 1414, w którym używany jest system SYSTEM.DEF.SVRCONN, który łączy się za pomocą transportu klienta.
- Kolejka o nazwie Q1.

Procedura

1. Uruchom program powłoki serwera GlassFish Server **asadmin** .
   a) Otwórz wiersz komend Windows i przejdź do katalogu *GlassFish*/bin , gdzie *GlassFish* to katalog, w którym zainstalowano serwer GlassFish Server w wersji 4.
   b) Wprowadź komendę **asadmin** w wierszu komend.

Komenda **asadmin** otwiera program powłoki w wierszu komend, który umożliwia utworzenie nowej domeny.

Serwer GlassFish Server w wersji 4 został uruchomiony w systemie.

2. Utwórz domenę, a następnie uruchom domenę.
   a) Aby utworzyć nową domenę, należy użyć komendy **create-domain** , podając nazwę portu i domeny. W wierszu komend wpisz następującą komendę:

   ```
   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 powiązanych z tym komendą. Jednak w przypadku tego zadania wymagany jest tylko parametr `-- adminport` . Więcej informacji na ten temat można znaleźć w dokumentacji produktu GlassFish Server 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 obecnie niedostępne.

   b) Po wyświetleniu monitu o wprowadzenie nazwy użytkownika i hasła wprowadź referencje, które mają być używane do logowania się na serwerze aplikacji za pomocą przeglądarki WWW.

   Jeśli wykonanie komendy zakończy się pomyślnie, w wierszu komend zostanie wyświetlony komunikat podsumowujący tworzenie domeny, w tym komunikat `Command create-domain executed successfully.` .

   Domena została pomyślnie utworzona.

   c) Uruchom domenę, wprowadzając następującą komendę w wierszu komend:

   ```
   start-domain domain_name
   ```

   gdzie *nazwa_domeny* jest nazwą domeny, która została wcześniej określona.

3. Aby uzyskać dostęp do serwera aplikacji GlassFish , należy użyć przeglądarki WWW.
   a) Na pasku adresu przeglądarki WWW wprowadź następującą komendę:

   ```
   localhost:port
   ```

   gdzie *port* to port podany wcześniej podczas tworzenia domeny.

   Zostanie wyświetlona konsola GlassFish .

   b) Po załadowaniu konsoli GlassFish Console użytkownik zostanie poproszony o podanie nazwy użytkownika i hasła, a następnie 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 **Deploy** (Wdrażaj), aby otworzyć stronę **Deploy Applications or Modules** (Wdrażanie aplikacji lub modułów).

   c) Kliknij przycisk **Przeglądaj** , a następnie przejdź do położenia pliku `wmq.jmsra.rar` . Wybierz ten plik i kliknij przycisk **OK**.

5. Utwórz pulę połączeń.
   a) Na pasku narzędzi, w obszarze **Zasoby**, wybierz pozycję menu **Konektory** .

   b) Następnie należy wybrać opcję menu **Pule połączeń konektora** , aby otworzyć stronę **Pule połączeń konektora** .

c) Kliknij przycisk **Nowy** , aby otworzyć stronę **Nowa pula połączeń konektora (krok 1 z 2)**).

d) Na stronie **Nowa pula połączeń konektora (krok 1 z 2)** należy wprowadzić nazwę puli jako jms/ivt/IVTCF-Connection-Pool w polu **Nazwa puli** .

e) W polu **Resource Adapter** (Adapter zasobów) wybierz plik **wmq.jmsra**.

f) W polu **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 puste wszystkie pozostałe pola.

f) Dla każdej z następujących par właściwości/wartość kliknij opcję **Dodaj właściwość**, a następnie wprowadź nazwę właściwości i wartość, tak jak przedstawiono to w poniższym przykładzie:

- nazwa: host; wartość: localhost
- nazwa: port; wartość 1414
- nazwa: kanał; wartość: SYSTEM.DEF.SVRCONN
- name: queueManager; wartość: QM
- name: transportType; wartość: CLIENT

Uwaga: Należy upewnić się, że używane są poprawne wartości dla własnych ustawień konfiguracyjnych, które mogą być inne niż te, które zostały przedstawione w tym przykładzie.

g) Na pasku narzędzi, w sekcji **Konektory**, wybierz opcję menu **Zasoby obiektu administracyjnego** , aby otworzyć stronę **Zasoby obiektu administracyjnego** .

h) Na stronie **Admin Object Resources** (Zasoby obiektu administracyjnego) kliknij opcję **New** (Nowy), aby otworzyć stronę **New Admin Object Resource** (Nowy zasób obiektu administracyjnego)

i) W polu **Nazwa JNDI** wprowadź wartość `IVTQueue`.

j) W polu **Resource Adapter** (Adapter zasobów) wpisz `wmq.jmsra`.

k) W polu **Resource Type** (Typ zasobu) wpisz `javax.jms.Queue`.

l) Pozostaw pole **Nazwa klasy** w taki sposób, w jakim jest.

m) Dla każdej z następujących par właściwości/wartość kliknij opcję **Dodaj właściwość**, a następnie wprowadź nazwę właściwości i wartość, tak jak przedstawiono to w poniższym przykładzie:

- nazwa: nazwa; wartość: IVTQueue
- nazwa: baseQueueManagerName; value QM
- nazwa: baseQueueNazwa; wartość: Q1

Uwaga: Należy upewnić się, że używane są poprawne wartości dla własnych ustawień konfiguracyjnych, które mogą być inne niż te, które zostały 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 Server.

a) Kliknij opcję **Aplikacje** na pasku narzędzi, aby wyświetlić stronę **Aplikacje** .

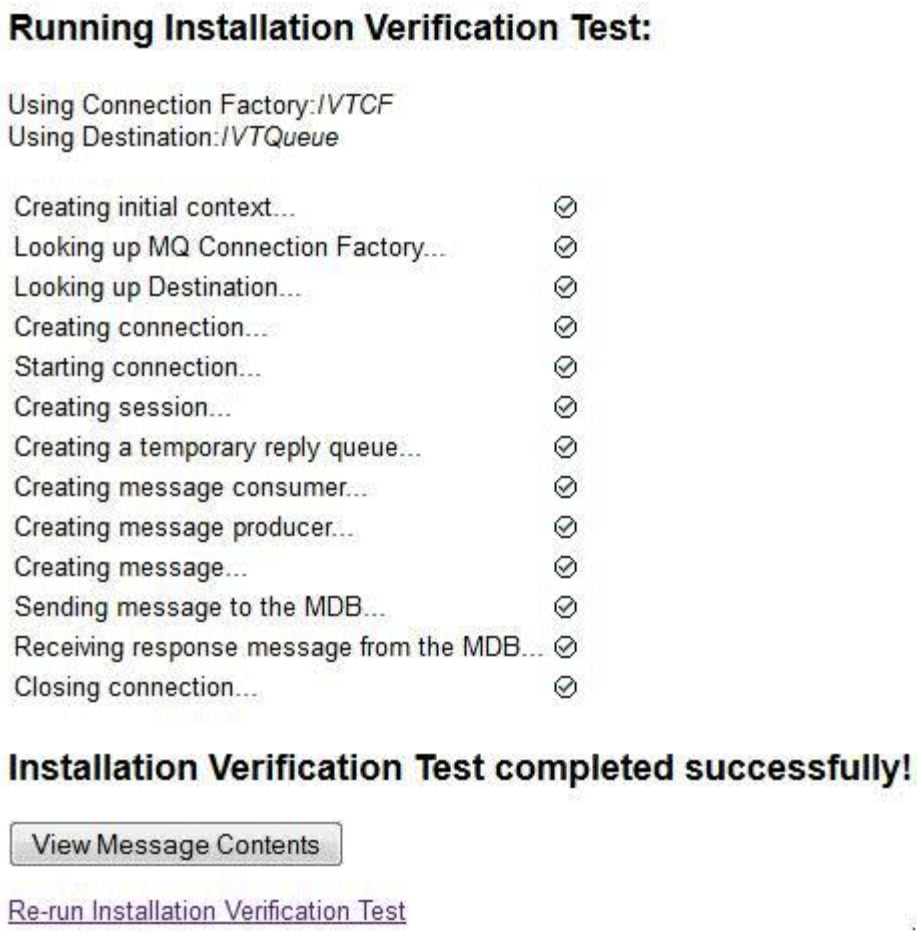
b) Kliknij opcję **Wdróż** , aby dodać aplikację IVT.

c) In the **Położenie** field navigate to, and select, the `wmq.jmsra.ivt.ear`.

d) W polu **Virtual Servers** (Serwery wirtualne) wybierz opcję **server**(Serwer), a następnie kliknij przycisk **OK**.

8. Uruchom program IVT.
   a) Kliknij opcję **Aplikacje** na pasku narzędzi, aby wyświetlić stronę **Aplikacje** .
   b) Kliknij opcję `wmq.jmsra.ivt` w tabeli Wdrożone aplikacje.
   c) Kliknij przycisk **Uruchom** w tabeli Moduły i komponenty.
   d) Wybierz odsyłacz http: link.
   e) Kliknij opcję **Uruchom IVT**.

   Uruchomiono program IVT, a jeśli się powiedzie, wyświetlane są następujące dane wyjściowe:



Rysunek 55. Pomyślne wyjście programu IVT

Instalowanie i testowanie adaptera zasobów w WildFly

Jeśli adapter zasobów produktu IBM MQ jest instalowany w środowisku WildFly V10, należy najpierw dokonać zmian w pliku konfiguracyjnym, aby dodać definicję podsystemu dla adaptera zasobów produktu IBM MQ . Następnie można wdrożyć adapter zasobów i przetestować go, instalując i uruchamiając aplikację testową weryfikacyjną instalacji (IVT).

O tym zadaniu

Ważne: Te instrukcje dotyczą WildFly V10.

W przypadku tego zadania założono, że użytkownik korzysta z działającego serwera aplikacji WildFly , a użytkownik zapoznał się ze standardowymi zadaniami administracyjnymi. W tym zadaniu założono również, że użytkownik dysponuje instalacją produktu IBM MQ i jest zaznajomiona ze standardowymi zadaniami administracyjnymi.

Procedura

1. Utwórz menedżer kolejek produktu IBM MQ o nazwie ExampleQMi ustaw go zgodnie z opisem w sekcji “Konfigurowanie menedżera kolejek w celu akceptowania połączeń klientów na wielu platformach” na stronie 1177.

   Podczas konfigurowania menedżera kolejek należy zwrócić uwagę na następujące punkty:

   - Nasłuchiwanie musi być uruchomione na porcie 1414.
   - Kanał, który ma być używany, nosi nazwę SYSTEM.DEF.SVRCONN.
   - Nazwa kolejki używana przez aplikację IVT nosi nazwę TEST.QUEUE.

   Kolejka modelowa SYSTEM.DEFAULT.MODEL.QUEUE musi również mieć nadane uprawnienia DSP i PUT, aby aplikacja mogła utworzyć tymczasową kolejkę odpowiedzi.

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. Przeprowadź wdrożenie adaptera 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, wyświetlając wiersz komend, przechodzący do katalogu *`WildFly_Home`*`/bin` i uruchamiający komendę:

```
standalone.bat -c standalone-full.xml
```

6. Uruchom aplikację IVT.

   Więcej informacji na ten temat zawiera "Weryfikowanie instalacji adaptera zasobów" na stronie 498. W przypadku WildFlydomyślnym adresem URL jest `http://localhost:8080/WMQ_IVT/`.

Używanie produktów IBM MQ i WebSphere Application Server razem

Through the IBM MQ messaging provider in WebSphere Application Server, Java Message Service (JMS) messaging applications can use your IBM MQ system as an external provider of JMS messaging resources.

O tym zadaniu

Aplikacje napisane w produkcie Java , które działają pod kontrolą produktu WebSphere Application Server , mogą używać specyfikacji Java Messaging Service (JMS) do przesyłania komunikatów. Przesyłanie komunikatów w tym środowisku może być udostępniane przez menedżera kolejek produktu IBM MQ .

Zaletą korzystania z menedżera kolejek produktu IBM MQ jest to, że łączenie aplikacji JMS może w pełni uczestniczyć w funkcjonalności sieci IBM MQ , która umożliwia aplikacjom wymianę komunikatów z menedżerami kolejek, które działają w wielu różnych platformach.

Aplikacje mogą korzystać z *transportu klienta* lub *transportu powiązań* dla obiektu fabryki połączeń kolejki. W przypadku transportu powiązań menedżer kolejek musi istnieć lokalnie w aplikacji, która wymaga połączenia.

Domyślnie komunikaty produktu JMS , które są przechowywane w kolejkach produktu IBM MQ , używają nagłówka MQRFH2 do przechowywania niektórych informacji nagłówka komunikatu produktu JMS . Wiele wcześniejszych aplikacji produktu IBM MQ nie może przetwarzać komunikatów z tymi nagłówkami i wymagać ich własnych, charakterystycznych nagłówków, na przykład MQCIH for CICS Bridge, lub MQWIH dla aplikacji IBM MQ Workflow. Więcej informacji na temat tych szczególnych uwag zawiera sekcja Odwzorowanie komunikatów produktu JMS na komunikaty produktu IBM MQ.

Zadania pokrewne

Konfigurowanie zasobów produktu JMS w produkcie WebSphere Application Server

Konfigurowanie serwera aplikacji pod względem używania najnowszego poziomu konserwacyjnego adaptera zasobów

Używanie produktu WebSphere Application Server z produktem IBM MQ

Produkty IBM MQ i IBM MQ for z/OS mogą być używane razem z domyślnym dostawcą przesyłania komunikatów lub jako alternatywa dla domyślnego dostawcy przesyłania komunikatów dołączonego do produktu WebSphere Application Server.

Dostawca przesyłania komunikatów produktu IBM MQ jest instalowany jako część produktu WebSphere Application Server. Dotyczy to wersji adaptera zasobów produktu IBM MQ oraz funkcji Extended Transactional Client produktu IBM MQ , która umożliwia menedżerowi kolejek uczestniczenie w transakcjach XA zarządzanych przez serwer aplikacji. Przy użyciu adaptera zasobów komponenty bean sterowane komunikatami można skonfigurować w taki sposób, aby używały specyfikacji aktywowania lub portów nasłuchiwania.

Aby serwer aplikacji był obsługiwany, należy wdrożyć program testowy instalacji adaptera zasobów produktu IBM MQ na serwerze aplikacji i uruchomić go pomyślnie. Po pomyślnym uruchomieniu programu testowego sprawdzania poprawności instalacji adaptera zasobów produktu IBM MQ adapter zasobów produktu IBM MQ może nawiązać połączenie z dowolnym obsługiwanym menedżerem kolejek produktu IBM MQ .

Połączenia JMS z WebSphere Application Server do IBM MQ

Przed rozważyniem poziomów produktu IBM MQ , które mogą być używane z produktem WebSphere Application Server, należy zrozumieć, w jaki sposób aplikacje Java Message Service (JMS) działające wewnątrz serwera aplikacji mogą łączyć się z menedżerami kolejek produktu IBM MQ .

Aplikacje produktu JMS , które muszą mieć dostęp do zasobów menedżera kolejek produktu IBM MQ , mogą to zrobić, korzystając z jednego z następujących typów transportu:

POWIĄZANIA
Ten transport może być używany, gdy serwer aplikacji i menedżer kolejek są zainstalowane na tym samym komputerze i na tym samym obrazie systemu operacyjnego. W przypadku korzystania z trybu BINDINGS cała komunikacja między tymi dwoma produktami odbywa się za pomocą komunikacji międzyprocesowej (Inter-Process Communication-IPC).

Dostawca przesyłania komunikatów produktu IBM MQ nie zawiera bibliotek rodzimych wymaganych do nawiązania połączenia z menedżerem kolejek produktu IBM MQ w trybie powiązań BINDINGS. Aby można było korzystać z połączenia w trybie BINDINGS, produkt IBM MQ musi być zainstalowany na tym samym komputerze, na którym znajduje się serwer aplikacji, a ścieżka biblioteki rodzimej 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 :

- Informacje na temat produktu WebSphere Application Server traditionalzawiera sekcja Konfigurowanie dostawcy przesyłania komunikatów produktu IBM MQ przy użyciu bibliotek rodzimych.
- Informacje na temat produktu WebSphere Libertyzawiera sekcja Wdrażanie aplikacji JMS na serwerze Liberty w celu używania dostawcy przesyłania komunikatów produktu IBM MQ.

z/OS W systemie z/OS, jeśli fabryka połączeń produktu WebSphere Application Server ma być połączona z menedżerem kolejek produktu IBM MQ w trybie powiązań, należy określić poprawne biblioteki produktu IBM MQ w konkatenacji STEPLIB produktu WebSphere Application Server . Więcej informacji na ten temat zawiera sekcja BibliotekiIBM MQ i WebSphere Application Server biblioteki STEPLIB produktu z/OS w dokumentacji produktu WebSphere Application Server .

KLIENT
Transport klienta używa protokołu TCP/IP do komunikacji między WebSphere Application Server a IBM MQ. Podobnie jak w przypadku, gdy serwer aplikacji i menedżer kolejek znajdują się na różnych komputerach, tryb CLIENT może być również używany, gdy dwa produkty są zainstalowane na tym samym komputerze i na tym samym obrazie systemu operacyjnego.

Aplikacje produktu JMS mogą również określać typ transportu BINDINGS_THEN_CLIENT. Gdy używany jest ten typ transportu, aplikacja będzie początkowo próbowała połączyć się z menedżerem kolejek przy użyciu trybu BINDINGS-jeśli nie jest w stanie tego zrobić, spróbuje wykonać transport CLIENT.

Jak znaleźć wersję adaptera zasobów produktu IBM MQ , który jest zainstalowany w produkcie WebSphere Application Server

Informacje o tym, która wersja adaptera zasobów produktu IBM MQ jest zainstalowana w produkcie WebSphere Application Server, zawiera nota techniczna Która wersja adaptera zasobów WebSphere MQ (RA) jest dostarczana z produktem WebSphere Application Server?.

W celu określenia poziomu 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: Aby uruchomić tę komendę, należy dwukrotnie kliknąć opcję **Wróć** po wprowadzeniu tej komendy.

JACL

```
set wmqInfoMBeans [$AdminControl queryNames WebSphere:type=WMQInfo,*]
foreach wmqInfoMBean $wmqInfoMBeans {
puts $wmqInfoMBean;
puts [$AdminControl invoke $wmqInfoMBean getInfo [] []]
}
```

Aktualizowanie adaptera zasobów

Aktualizacje adaptera zasobów produktu IBM MQ , które są instalowane razem z serwerem aplikacji, są dołączone do pakietów poprawek produktu WebSphere Application Server . Aktualizowanie adaptera zasobów produktu IBM MQ przy użyciu opcji **Aktualizuj adapter zasobów ...** Narzędzie w Konsoli administracyjnej produktu WebSphere Application Server nie jest zalecane, ponieważ oznacza to, że aktualizacje udostępnione w pakiecie poprawek produktu WebSphere Application Server nie będą miały żadnego wpływu.

MQ_INSTALL_ROOT, zmienna

W wersjach wcześniejszych niż 7.0produkt WebSphere Application Server można skonfigurować w taki sposób, aby używany był serwer IBM WebSphere MQ classes for JMS znajdujący się w zewnętrznej instalacji produktu IBM WebSphere MQ w celu nawiązania połączenia z menedżerem kolejek. W tym celu należy ustawić zmienną MQ_INSTALL_ROOT dla zmiennej WebSphere .

Z poziomu produktu WebSphere Application Server 7.0wartość MQ_INSTALL_ROOT jest używana tylko do znajdowania bibliotek rodzimych i jest przesłaniana przez dowolną ścieżkę do biblioteki rodzimej skonfigurowaną na adapterze zasobów.

Nawiąże połączenie z WebSphere Application Server do IBM MQ

Ostrzeżenie:

1. Dowolna obsługiwana wersja produktu WebSphere Application Server może korzystać z adaptera zasobów produktu IBM MQ , który jest z nim spakowany, w celu nawiązania połączenia z dowolną obsługiwaną wersją produktu IBM MQ.
2. Jeśli używany jest tryb powiązań, niektóre biblioteki w programie WebSphere Application Server muszą być zgodne z wersją menedżera kolejek, z którym łączy się:
   - Produkt WebSphere Application Server musi być skonfigurowany w taki sposób, aby ładował biblioteki rodzime udostępnione w produkcie IBM MQ 9.1. Więcej informacji zawiera sekcja “Konfigurowanie bibliotek rodzimego interfejsu produktu Java (JNI)” na stronie 93.
   - z/OS W systemie z/OSnależy określić poprawne biblioteki produktu IBM MQ w konkatenacji produktu WebSphere Application Server STEPLIB.

     Szczegółowe informacje na temat wymaganych bibliotek produktu IBM MQ znajdują się w sekcji BibliotekiIBM MQ i biblioteki STEPLIB produktu WebSphere Application Server for z/OS .

Jeśli w systemie LINKLIST (LINKLIST-LINKLIST) znajdują się biblioteki dla jednej wersji produktu IBM MQ , można połączyć się z inną wersją produktu IBM MQ , nadpisując biblioteki w bibliotece STEPLIB.

3. Wersja adaptera zasobów produktu IBM MQ jest niezależna od rodzimych (współużytkowanych) wersji biblioteki udostępnianych przez instalację menedżera kolejek.

   Na przykład WebSphere Application Server 8.5 z adapterem zasobów IBM WebSphere MQ 7.1 może nadal zarządzać powiązaniami powiązań z menedżerem kolejek produktu IBM MQ 9.0 przy użyciu rodzimych bibliotek produktu IBM MQ 9.0 .

Więcej informacji na ten temat zawiera “Obsługa instrukcji adaptera zasobów IBM MQ” na stronie 444.

W poniższej tabeli przedstawiono typy transportu, które mogą zostać użyte do nawiązania połączenia z produktem IBM MQ ze wszystkich wersji produktu WebSphere Application Server.

Wersja produktu IBM MQ lub IBM WebSphere MQ	**transport powiązań**	**Transport KLIENTA**
IBM MQ 9.1	Obsługiwane. • Produkt IBM MQ 9.1 musi być zainstalowany na tym samym komputerze, na którym znajduje się serwer aplikacji. • Produkt WebSphere Application Server musi być skonfigurowany w taki sposób, aby ładował biblioteki rodzime udostępnione w produkcie IBM MQ 9.1. • z/OS W systemie z/OS, jeśli fabryka połączeń produktu WebSphere Application Server ma być połączona z menedżerem kolejek produktu IBM MQ w trybie powiązań, w konkatenacji STEPLIB produktu WebSphere Application Server należy określić poprawne biblioteki produktu IBM MQ .	Obsługiwany
IBM MQ 9.0	Obsługiwane. • Produkt IBM MQ 9.0 musi być zainstalowany na tym samym komputerze, na którym znajduje się serwer aplikacji. • Produkt WebSphere Application Server musi być skonfigurowany w taki sposób, aby ładował biblioteki rodzime udostępnione w produkcie IBM MQ 9.0. • z/OS W systemie z/OS, jeśli fabryka połączeń produktu	Obsługiwany

Wersja produktu IBM MQ lub IBM WebSphere MQ	transport powiązań	Transport KLIENTA
	WebSphere Application Server ma być połączona z menedżerem kolejek produktu IBM MQ w trybie powiązań, w konkatenacji STEPLIB produktu WebSphere Application Server należy określić poprawne biblioteki produktu IBM MQ .	
IBM MQ 8.0	Obsługiwane. • Produkt IBM MQ 8.0 musi być zainstalowany na tym samym komputerze, na którym znajduje się serwer aplikacji. • Produkt WebSphere Application Server musi być skonfigurowany w taki sposób, aby ładował biblioteki rodzime udostępnione w produkcie IBM MQ 8.0. • z/OS W systemie z/OS, jeśli fabryka połączeń produktu WebSphere Application Server ma być połączona z menedżerem kolejek produktu IBM MQ w trybie powiązań, w konkatenacji STEPLIB produktu WebSphere Application Server należy określić poprawne biblioteki produktu IBM MQ .	Obsługiwany
IBM WebSphere MQ 7.5	Obsługiwane. • Produkt IBM WebSphere MQ 7.5 musi być zainstalowany na tym samym komputerze, na którym znajduje się serwer aplikacji. • Produkt WebSphere Application Server musi być skonfigurowany w taki sposób, aby ładował biblioteki rodzime udostępnione w produkcie IBM WebSphere MQ 7.5. • z/OS W systemie z/OS, jeśli fabryka połączeń produktu WebSphere Application Server ma zostać połączona z menedżerem kolejek	Obsługiwany

Wersja produktu IBM MQ lub IBM WebSphere MQ	transport powiązań	Transport KLIENTA
	produktu IBM WebSphere MQ w trybie powiązań, należy określić poprawne biblioteki produktu IBM WebSphere MQ w konkatenacji STEPLIB produktu WebSphere Application Server .	
IBM WebSphere MQ 7.1	Obsługiwane. • Produkt IBM WebSphere MQ 7.1 musi być zainstalowany na tym samym komputerze, na którym znajduje się serwer aplikacji. • Produkt WebSphere Application Server musi być skonfigurowany w taki sposób, aby ładował biblioteki rodzime udostępnione w produkcie IBM WebSphere MQ 7.1. • z/OS W systemie z/OS, jeśli fabryka połączeń produktu WebSphere Application Server ma zostać połączona z menedżerem kolejek produktu IBM WebSphere MQ w trybie powiązań, należy określić poprawne biblioteki produktu IBM WebSphere MQ w konkatenacji STEPLIB produktu WebSphere Application Server .	Obsługiwany

W poniższej tabeli przedstawiono wersje produktu WebSphere Application Server , w których jest obsługiwany adapter zasobów IBM MQ .

Wersja adaptera zasobów produktu IBM MQ	Która wersja produktu WebSphere Application Server może być uruchomiona w tej wersji adaptera zasobów?
IBM MQ 9.1	Adapter zasobów może działać w: • Any Java EE 7 compliant version of WebSphere Liberty. • WebSphere Application Server traditional 9.0.
IBM MQ 9.0	Adapter zasobów może działać w: • Any Java EE 7 compliant version of WebSphere Liberty. • WebSphere Application Server traditional 9.0

Wersja adaptera zasobów produktu IBM MQ	Która wersja produktu WebSphere Application Server może być uruchomiona w tej wersji adaptera zasobów?
IBM MQ 8.0	Adapter zasobów może działać w dowolnej wersji produktu WebSphere Liberty zgodnej z produktem Java EE 7 . Adapter zasobów IBM MQ 8.0 nie jest obsługiwany w celu uruchomienia w produkcie WebSphere Application Server traditional. Adapter zasobów, który jest już zainstalowany w produkcie WebSphere Application Server traditional , powinien być używany do nawiązywania połączenia z menedżerami kolejek produktu IBM MQ 8.0 .
IBM WebSphere MQ 7.5	Adapter zasobów może być używany na serwerach aplikacji zgodnych z J2EE 1.4 lub nowszymi. Wersja adaptera zasobów produktu IBM WebSphere MQ zawarta w produkcie WebSphere Application Server 8.0 i 7.0 powinna być używana w tych środowiskach. Produkty IBM WebSphere MQ 7.5.0 Fix Pack 2 i APAR IC92914 dodają obsługę wdrażania adaptera zasobów w produkcie WebSphere Liberty.
IBM WebSphere MQ 7.1	Adapter zasobów może być używany na serwerach aplikacji zgodnych z J2EE 1.4 lub nowszymi. Wersja adaptera zasobów produktu IBM WebSphere MQ zawarta w produkcie WebSphere Application Server 8.0 i 7.0 powinna być używana w tych środowiskach.

Pojęcia pokrewne

“Obsługa instrukcji adaptera zasobów IBM MQ” na stronie 444
Adapter zasobów dostarczany z produktem IBM MQ 8.0 lub nowszym 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 w związku z tym obsługuje produkt JMS 2.0.

Informacje pokrewne

Wymagania systemowe produktu IBM MQ

Określanie liczby połączeń TCP/IP, które są tworzone z WebSphere Application Server do IBM MQ

Za pomocą funkcji współużytkowania konwersacji wiele konwersacji może współużytkować instancje kanału MQI, co jest również znane jako połączenie TCP/IP.

O tym zadaniu

Aplikacje działające w produkcie WebSphere Application Server 7 i 8, które korzystają z trybu normalnego dostawcy przesyłania komunikatów produktu IBM MQ , automatycznie korzystają z tej funkcji. Oznacza to, że wiele aplikacji działających w obrębie tej samej instancji serwera aplikacji, które łączą się z tym samym menedżerem kolejek produktu IBM MQ , są w stanie współużytkować tę samą instancję kanału.

Liczba konwersacji, które mogą być współużytkowane przez pojedynczą instancję kanału, jest określana przez właściwość kanału IBM MQ **SHARECNV**. Domyślna wartość tej właściwości dla kanałów połączenia z serwerem wynosi 10.

Patrząc na liczbę konwersacji tworzonych przez produkt WebSphere Application Server 7 i 8 możliwe jest określenie liczby tworzonych instancji kanału.

Więcej informacji na temat trybu dostawcy przesyłania komunikatów produktu IBM MQ zawiera sekcja Tryb normalny dostawcy.

Pojęcia pokrewne

Korzystanie z współużytkowanych konwersacji
W środowisku, w którym dozwolone jest współużytkowanie konwersacji, konwersacje mogą współużytkować instancję kanału MQI.

“Współużytkowanie połączenia TCP/IP w produkcie IBM MQ classes for JMS” na stronie 313
W celu współużytkowania pojedynczego połączenia 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, a także dla każdej sesji JMS utworzonej z połączenia JMS .

Jedna konwersacja dla każdego połączenia JMS , które zostało utworzone z fabryki połączeń.

Każda fabryka połączeń produktu JMS ma powiązaną pulę połączeń, która jest podzielona na dwie sekcje: wolną pulę i aktywną pulę. Oba pule są początkowo puste.

Gdy aplikacja tworzy połączenie produktu JMS z fabryki połączeń, program WebSphere Application Server sprawdza, czy w puli wolnych jest połączenie JMS . Jeśli jest, jest przenoszony do aktywnej puli i jest podawany do aplikacji. W przeciwnym razie zostanie utworzone nowe połączenie z produktem JMS , które zostanie umieszczone w aktywnej puli i zostanie 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 przy użyciu połączenia z produktem JMS i zamknięciu połączenia, połączenie jest przenoszone z aktywnej puli do puli wolnych miejsc, gdzie jest ona dostępna do ponownego wykorzystania. Właściwość puli połączeń **Unused timeout** określa, jak długo połączenie z produktem JMS może pozostać w wolnej puli, zanim zostanie rozłączone. Wartość domyślna tej właściwości to 1800 sekund (30 minut).

Po pierwszym utworzeniu połączenia JMS rozpoczyna się konwersacja między programem WebSphere Application Server i produktem IBM MQ . Konwersacja pozostaje aktywna do momentu zamknięcia połączenia, gdy wartość właściwości **Unused timeout** dla wolnej puli zostanie przekroczona.

Jedna konwersacja dla każdej sesji JMS , która została utworzona z połączenia JMS .

Każde połączenie JMS utworzone z fabryki połączeń dostawcy przesyłania komunikatów produktu IBM MQ ma powiązaną pulę sesji JMS . Pule sesji działają w taki sam sposób, jak pule połączeń. Maksymalna liczba sesji JMS , które można utworzyć z pojedynczego połączenia JMS , jest określana przez właściwość puli sesji fabryki połączeń **Maximum connections**. Wartością domyślną tej właściwości jest 10.

Konwersacja jest uruchamiana po pierwszym utworzeniu sesji JMS . konwersacja pozostaje aktywna do momentu zamknięcia sesji JMS , ponieważ pozostała w puli wolnych od wartości właściwości **Unused timeout** dla puli sesji.

Obliczanie wartości dla właściwości SHARECNV

Maksymalną liczbę konwersacji z pojedynczej fabryki połączeń można obliczyć na IBM MQ przy użyciu następującego wzoru:

```
Maximum number of conversations =
    connection Pool Maximum Connections +
   (connection Pool Maximum Connections * Session Pool Maximum Connections)
```

Liczba instancji kanałów, które zostaną utworzone w celu umożliwienia wykonania tej liczby konwersacji, można wypracować przy użyciu następującego obliczenia:

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

Każda pozostała część z tego obliczenia może zostać zaokrąglona w górę.

W przypadku prostej fabryki połączeń, która używa wartości domyślnej dla puli połączeń **Maximum connections** i właściwości puli sesji **Maximum connections** , maksymalna liczba konwersacji, które mogą istnieć między WebSphere Application Server a IBM MQ dla tej fabryki połączeń, to:

```
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 łączy się z produktem IBM MQ przy użyciu kanału, który ma właściwość **SHARECNV** ustawioną na wartość 10, to maksymalna liczba instancji kanału, które zostaną utworzone dla tej fabryki połączeń, to:

```
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 pod kątem 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 bean sterowanych komunikatami, które są skonfigurowane pod kątem używania specyfikacji aktywowania:

- Jedna konwersacja dla specyfikacji aktywowania w celu 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.

Właściwość zaawansowana specyfikacji aktywowania **Maximum server sessions** określa maksymalną liczbę sesji serwera, które mogą być aktywne w dowolnym momencie dla danej specyfikacji aktywowania. Ta właściwość ma wartość domyślną 10. Sesje serwera są tworzone w miarę potrzeb i są zamykane, jeśli były bezczynne przez okres określony w zaawansowanej właściwości specyfikacji aktywowania **Server session pool timeout**. Wartość domyślna tej właściwości to 300000 milisekund (5 minut).

Konwersacje są uruchamiane podczas tworzenia sesji serwera i są zatrzymane po zatrzymaniu specyfikacji aktywowania lub w momencie, gdy sesja serwera jest przerwa.

Oznacza to, że maksymalna liczba konwersacji z pojedynczej specyfikacji aktywowania do produktu IBM MQ może być obliczona przy użyciu następującej formuły:

```
Maximum number of conversations = Maximum server sessions + 1
```

Liczba instancji kanału, które są tworzone w celu umożliwienia wykonania tej liczby konwersacji, można znaleźć, wykonując następujące obliczenia:

```
Maximum number of channel instances =
    Maximum number of conversations / SHARECNV for the channel being used
```

Każda pozostała część 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 WebSphere Application Server a IBM MQ dla tej specyfikacji aktywowania, jest obliczana jako:

```
Maximum number of conversations = Maximum server sessions + 1
```

Na przykład:

```
    = 10 + 1
    = 11
```

Jeśli ta specyfikacja aktywowania łączy się z produktem IBM MQ przy użyciu kanału, który ma właściwość **SHARECNV** ustawioną na wartość 10, wówczas liczba tworzonych instancji kanałów 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 używane przez aplikacje komponentu bean sterowanego komunikatami tworzą konwersacje dla każdej sesji serwera. Jeden z nich monitoruje miejsce docelowe dla odpowiednich komunikatów, a inny uruchamia instancję komponentu bean sterowanego komunikatami w celu przetwarzania komunikatów. Liczba konwersacji dla każdego portu nasłuchiwania może być obliczona na podstawie maksymalnej liczby sesji.

Domyślnie porty nasłuchiwania będą uruchamiane w trybie ASF jako część specyfikacji 1.1 , która definiuje mechanizm, którego serwery aplikacji powinny używać do wykrywania komunikatów i dostarczania ich do komponentów bean sterowanych komunikatami w celu ich przetwarzania. Aplikacje komponentu bean sterowanego komunikatami, które są skonfigurowane do używania portów nasłuchiwania w tym domyślnym trybie operacji, tworzą konwersacje:

Jedna konwersacja dla portu nasłuchiwania w celu monitorowania miejsca docelowego dla odpowiednich komunikatów.

Porty nasłuchiwania są skonfigurowane do korzystania z fabryki połączeń produktu JMS . Po uruchomieniu portu nasłuchiwania żądanie jest wykonywane dla połączenia JMS z wolnej puli fabryki połączeń. Połączenie zostanie zwrócone do puli wolnych, gdy port nasłuchiwania zostanie zatrzymany. Więcej informacji na temat sposobu korzystania z puli połączeń oraz sposobu, w jaki wpływa to na liczbę konwersacji w programie IBM MQ, zawiera sekcja “Fabryki połączeń usługi JMS” na stronie 513.

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 pobranych z puli sesji powiązanej z połączeniem JMS , z którego korzysta port nasłuchiwania.

Jeśli sesja serwera jest bezczynna przez okres określony przez właściwość niestandardową usługi nasłuchiwania komunikatów **SERVER.SESSION.POOL.UNUSED.TIMEOUT**, sesja jest zamknięta, a używana sesja JMS jest zwracana do puli wolnej puli sesji. Sesja JMS pozostanie w puli wolnych pul sesji, dopóki nie będzie potrzebna lub jest zamknięta, ponieważ w puli wolnej od wartości dłuższej niż wartość właściwości **Unused timeout** puli sesji jest bezczynna.

Więcej informacji na temat sposobu korzystania z puli sesji oraz sposobu zarządzania konwersacjami między produktem WebSphere Application Server i produktem IBM MQ zawiera sekcja “Fabryki połączeń usługi JMS” na stronie 513.

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 IBM MQ

Maksymalną liczbę konwersacji z jednego portu nasłuchiwania można obliczyć na IBM MQ przy użyciu następującego wzoru:

```
Maximum number of conversations = Maximum sessions + 1
```

Liczba instancji kanałów, które zostaną utworzone w celu umożliwienia wykonania tej liczby konwersacji, można wypracować przy użyciu następującego obliczenia:

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

Każda pozostała część 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 WebSphere Application Server a IBM MQ dla tego portu nasłuchiwania, jest obliczana jako:

```
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, wówczas liczba instancji kanału, które zostaną utworzone, 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 bez ASF (Application Server Facilities)

Porty nasłuchiwania działające w trybie bez ASF można skonfigurować w taki sposób, aby monitorować miejsce docelowe kolejki i miejsce docelowe tematu przy użyciu sesji serwera. Sesje serwera mogą zawierać wiele konwersacji, których maksymalna liczba może być obliczona w każdym przypadku.

Porty nasłuchiwania można skonfigurować w taki sposób, aby były uruchamiane w trybie bez ASF, co zmienia sposób, w jaki porty nasłuchiwania monitorują miejsca docelowe produktu JMS . Aplikacje bean sterowane komunikatami, korzystające z portów nasłuchiwania w trybie bez ASF operacji, 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 w danym porcie nasłuchiwania. Wartością domyślną tej właściwości jest 10.

Podczas pracy w trybie bez ASF monitorowanie portu nasłuchiwania kolejki docelowej spowoduje automatyczne utworzenie liczby sesji serwera określonych przez właściwość portu nasłuchiwania **Maksymalna liczba sesji**. Wszystkie te sesje serwera korzystają z sesji JMS pobranych z puli sesji powiązanej z połączeniem JMS , z której korzysta port nasłuchiwania, i stale monitorują miejsce docelowe produktu JMS pod kątem odpowiednich komunikatów.

Jeśli port nasłuchiwania jest skonfigurowany do monitorowania miejsca docelowego tematu, wartość opcji **Maksymalna liczba sesji** jest ignorowana, a 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 sesje JMS , które zostały użyte, są zwracane do puli wolnych pul sesji dla połączenia JMS , z którego korzysta port nasłuchiwania.

Więcej informacji na temat sposobu korzystania z puli sesji oraz sposobu zarządzania konwersacjami między produktem WebSphere Application Server i produktem IBM MQ zawiera sekcja “Fabryki połączeń usługi JMS” na stronie 513.

Więcej informacji na temat trybu ASF i trybu bez ASF operacji z produktem WebSphere Application Serveroraz sposobu konfigurowania portów nasłuchiwania w celu korzystania z trybu bez ASF zawiera sekcja Przetwarzanie komunikatów w trybie ASF i w trybie bez ASF.

Obliczanie maksymalnej liczby konwersacji podczas monitorowania miejsca docelowego kolejki

Maksymalna liczba konwersacji z jednego portu nasłuchiwania, działających w trybie bez ASF i monitorowania miejsca docelowego kolejki do programu IBM MQ , może być obliczana przy użyciu następującej formuły:

```
Maximum number of conversations = Maximum sessions
```

Liczba instancji kanałów, które zostaną utworzone w celu umożliwienia wykonania tej liczby konwersacji, można znaleźć przy użyciu następującego obliczenia:

```
Maximum number of channel instances =
   Maximum number of conversations / SHARECNV for the channel being used
```

Każda pozostała część 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 WebSphere Application Server a IBM MQ dla tego portu nasłuchiwania to:

```
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, wówczas liczba tworzonych instancji kanału jest obliczana jako:

```
     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

W przypadku portu nasłuchiwania działającego w trybie bez ASF i skonfigurowanego do monitorowania miejsca docelowego tematu liczba konwersacji z portu nasłuchiwania do programu IBM MQ jest następująca:

```
Maximum number of conversations = 1
```

Liczba instancji kanałów, które zostaną utworzone w celu umożliwienia wykonania tej liczby konwersacji, można znaleźć przy użyciu następującego obliczenia:

```
     Maximum number of channel instances =
        Maximum number of conversations / SHARECNV for the channel being used
```

Każda pozostała część 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 WebSphere Application Server a IBM MQ dla tego portu nasłuchiwania to:

```
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, wówczas liczba tworzonych instancji kanału jest obliczana jako:

```
     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 WebSphere Application Server z produktem IBM MQ

Aliasy uwierzytelniania są odwzorowywać na kombinację nazwy użytkownika i hasła, która może być używana do zabezpieczania połączenia WebSphere Application Server z produktem IBM MQ. Fabrykę połączeń można skonfigurować przy użyciu aliasu uwierzytelniania.

Używanie aliasów uwierzytelniania z aplikacjami korporacyjnymi

Gdy aplikacja korporacyjna działająca w produkcie WebSphere Application Server podejmuje próbę utworzenia połączenia JMS z produktem IBM MQ, aplikacja wyszukuje definicję fabryki połączeń dostawcy przesyłania komunikatów produktu IBM MQ z repozytorium produktu Java Naming Directory Interface (JNDI) serwera aplikacji.

Gdy definicja fabryki połączeń dostawcy przesyłania komunikatów produktu IBM MQ znajduje się w repozytorium produktu JNDI serwera aplikacji, wywoływana jest jedna z następujących metod:

- `ConnectionFactory.createConnection()`
- `ConnectionFactory.createConnection(String username, String password)`

Jeśli fabryka połączeń została skonfigurowana przy użyciu zdefiniowanego aliasu uwierzytelniania J2C , nazwa użytkownika i hasło w aliasie uwierzytelniania mogą zostać włączone do produktu IBM MQ , gdy fabryka połączeń zostanie użyta do utworzenia połączenia.

Fabryki połączeń i aliasy uwierzytelniania

Fabryki połączeń dostawcy przesyłania komunikatów produktu IBM MQ zawierają informacje na temat nawiązywania połączenia z menedżerami kolejek produktu IBM MQ . Aplikacje korporacyjne działające w produkcie WebSphere Application Server mogą używać fabryk połączeń do tworzenia połączeń produktu JMS z produktem IBM MQ.

Produkt WebSphere Application Server przechowuje definicje fabryk połączeń w repozytorium, do którego można uzyskać dostęp za pomocą JNDI. Po utworzeniu fabryki połączeń fabryka połączeń otrzymuje nazwę JNDI , która jednoznacznie identyfikuje ją w zasięgu serwera aplikacji (komórka, węzeł lub zasięg serwera), na którym została zdefiniowana.

Na przykład fabryka połączeń dostawcy przesyłania komunikatów produktu IBM MQ zdefiniowana w zasięgu komórki produktu WebSphere Application Server zawiera informacje na temat sposobu łączenia się z menedżerem kolejek (`myQM`) przy użyciu transportu BINDINGS. Ta fabryka połączeń ma nazwę JNDI `jms/myCF` , która jednoznacznie identyfikuje ją.

Fabryki połączeń można również skonfigurować w taki sposób, aby używały aliasu uwierzytelniania. Aliasy uwierzytelniania są odwzorowywać na kombinację nazwy użytkownika i hasła. W zależności od sposobu użycia fabryki połączeń nazwa użytkownika i hasło w aliasie uwierzytelniania mogą być lub nie są bloczone do IBM MQ podczas tworzenia połączenia z produktem JMS .

Ważne: W wersjach wcześniejszych niż IBM MQ 8.0domyślny menedżer uprawnień do obiektu (Object Authority Manager-OAM) produktu IBM MQ przeprowadził kontrolę autoryzacji, aby upewnić się, że po nawiązaniu połączenia nazwa użytkownika przekazana do produktu IBM MQmiała uprawnienia dostępu do menedżera kolejek.

Nie dokonano żadnych sprawdzeń w celu sprawdzenia poprawności podanego hasła. Aby wykonać sprawdzenie uwierzytelniania i sprawdzić, czy identyfikator użytkownika i hasło są zgodne, konieczne jest zapisanie wyjścia zabezpieczeń kanału produktu IBM MQ . Szczegółowe informacje o tym, jak to zrobić, można znaleźć w sekcji Programy obsługi wyjścia zabezpieczeń kanału.

W produkcie IBM MQ 8.0menedżer kolejek sprawdza hasło oprócz nazwy użytkownika.

Korzystanie z fabryki połączeń

Poniższe tematy zawierają informacje na temat korzystania z fabryki połączeń przy użyciu bezpośrednich i pośrednich wyszukiwania:

- “Korzystanie z fabryki połączeń przy użyciu wyszukiwania bezpośredniego” na stronie 523
- “Korzystanie z fabryki połączeń przy użyciu wyszukiwania pośredniego” na stronie 524

Korzystanie z transportu CLIENT

Fabryki połączeń skonfigurowane do korzystania z transportu CLIENT muszą określać, który kanał połączenia z serwerem IBM MQ (SVRCONN) będzie używany do łączenia się z menedżerem kolejek.

Jeśli właściwość Identyfikator użytkownika agenta kanału IBM MQ (MCAUSER) pozostanie pusta dla kanału, który został skonfigurowany do użycia przez fabrykę połączeń, wówczas fabryka połączeń może być używana z bezpośrednim wyglądem lub z pośrednim wyglądnym wyglądem.

Jeśli właściwość MCAUSER jest ustawiona na identyfikator użytkownika, ten identyfikator użytkownika jest przekazywany do produktu IBM MQ , gdy fabryka połączeń jest używana do tworzenia połączenia z produktem IBM MQ, niezależnie od tego, czy aplikacja korporacyjna korzysta z bezpośredniego, czy pośredniego wyszukiwania.

tabele podsumowania

W poniższych tabelach podsumowano, które identyfikatory użytkowników są wyświetlane w dół do IBM MQ , gdy używany jest transport BINDINGS, a także transport CLIENT:

Tabela 69. 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 blokowy w dół do IBM MQ.	Identyfikator użytkownika i hasło, które zostały przekazane do metody `ConnectionFactory.createConnection(String username, String password)` , są przekazywane do produktu IBM MQ.
Deskryptor wdrażania aplikacji zawiera odwołanie do zasobu dla fabryki połączeń, a właściwość **`res-auth`** jest ustawiona na wartość "Aplikacja".	Identyfikator użytkownika dla procesu serwera aplikacji jest blokowy w dół do IBM MQ.	Identyfikator użytkownika i hasło, które zostały przekazane do metody `ConnectionFactory.createConnection(String username, String password)` , są przekazywane do produktu 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 dostępne w obszarze IBM MQ.	Identyfikator użytkownika i hasło określone w aliasie uwierzytelniania dla fabryki połączeń są dostępne w obszarze IBM MQ.

Tabela 69. 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 "Container", a aplikacja została skonfigurowana z aliasem uwierzytelniania.	Identyfikator użytkownika i hasło podane w aliasie uwierzytelniania, w którym aplikacja została skonfigurowana do użycia, są dostępne w obszarze IBM MQ.	Identyfikator użytkownika i hasło podane w aliasie uwierzytelniania, w którym aplikacja została skonfigurowana do użycia, są dostępne w obszarze IBM MQ.

Tabela 70. 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 pod kątem używania kanału produktu IBM MQ z nieustawioną właściwością MCAUSER.	Identyfikator użytkownika dla procesu serwera aplikacji jest blokowy w dół do IBM MQ.	Identyfikator użytkownika i hasło, które zostały przekazane do metody `ConnectionFactory.createConnection(String username, String password)`, są przekazywane do produktu IBM MQ.
Deskryptor wdrażania aplikacji nie zawiera odwołania do zasobu dla fabryki połączeń, a fabryka połączeń jest skonfigurowana pod kątem używania kanału produktu IBM MQ z właściwością MCAUSER ustawioną na identyfikator użytkownika.	Identyfikator użytkownika określony przez właściwość MCAUSER w kanale IBM MQ , którego fabryka połączeń jest skonfigurowana do użycia, jest bloczona w dół do IBM MQ.	Identyfikator użytkownika określony przez właściwość MCAUSER w kanale IBM MQ , którego fabryka połączeń jest skonfigurowana do użycia, jest bloczona w dół do 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 pod kątem używania kanału IBM MQ , który ma ustawioną właściwość MCAUSER.	Identyfikator użytkownika dla procesu serwera aplikacji jest blokowy w dół do IBM MQ.	Identyfikator użytkownika i hasło, które zostały przekazane do metody `ConnectionFactory.createConnection(String username, String password)`, są przekazywane do produktu IBM MQ.

Tabela 70. 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 pod kątem używania kanału produktu IBM MQ z właściwością MCAUSER ustawioną na identyfikator użytkownika.	Identyfikator użytkownika określony przez właściwość MCAUSER w kanale IBM MQ , którego fabryka połączeń jest skonfigurowana do użycia, jest przepływowa w dół do IBM MQ.	Identyfikator użytkownika określony przez właściwość MCAUSER w kanale IBM MQ , którego fabryka połączeń jest skonfigurowana do użycia, jest przepływowa w dół do IBM MQ.
Deskryptor wdrażania aplikacji zawiera odwołanie do zasobu dla fabryki połączeń, dla której właściwość **`res-auth`** jest ustawiona na wartość*Kontener* , a fabryka połączeń jest skonfigurowana pod kątem używania kanału produktu IBM MQ z nieustawioną właściwością MCAUSER.	Identyfikator użytkownika i hasło określone w aliasie uwierzytelniania dla fabryki połączeń są dostępne w obszarze IBM MQ.	Identyfikator użytkownika i hasło określone w aliasie uwierzytelniania dla fabryki połączeń są dostępne w obszarze IBM MQ.
Deskryptor wdrażania aplikacji zawiera odwołanie do zasobu dla fabryki połączeń, dla której właściwość **`res-auth`** jest ustawiona na wartość "*Kontener* , a fabryka połączeń jest skonfigurowana pod kątem używania kanału produktu IBM MQ z właściwością MCAUSER ustawioną na identyfikator użytkownika.	Identyfikator użytkownika określony przez właściwość MCAUSER w kanale IBM MQ , którego fabryka połączeń jest skonfigurowana do użycia, jest przepływowa w dół do IBM MQ.	Identyfikator użytkownika określony przez właściwość MCAUSER w kanale IBM MQ , którego fabryka połączeń jest skonfigurowana do użycia, jest przepływowa w dół do IBM MQ.
Deskryptor wdrażania aplikacji zawiera odwołanie do zasobu dla fabryki połączeń, dla której właściwość **`res-auth`** jest ustawiona na wartość*Kontener* , a aplikacja została skonfigurowana z aliasem uwierzytelniania, a fabryka połączeń jest skonfigurowana pod kątem używania kanału produktu IBM MQ z nieustawioną właściwością MCAUSER.	Identyfikator użytkownika i hasło podane w aliasie uwierzytelniania, w którym aplikacja została skonfigurowana do użycia, są dostępne w obszarze IBM MQ.	Identyfikator użytkownika i hasło podane w aliasie uwierzytelniania, w którym aplikacja została skonfigurowana do użycia, są dostępne w obszarze IBM MQ.

Tabela 70. 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 *Kontener* , a aplikacja została skonfigurowana z aliasem uwierzytelniania, a fabryka połączeń jest skonfigurowana pod kątem używania kanału produktu IBM MQ z ustawionym identyfikatorem użytkownika MCAUSER.	Identyfikator użytkownika określony przez właściwość MCAUSER w kanale IBM MQ , którego fabryka połączeń jest skonfigurowana do użycia, jest przepływowa w dół do IBM MQ.	Identyfikator użytkownika określony przez właściwość MCAUSER w kanale IBM MQ , którego fabryka połączeń jest skonfigurowana do użycia, jest przepływowa w dół do IBM MQ.

Korzystanie z fabryki połączeń przy użyciu wyszukiwania bezpośredniego

Po zdefiniowaniu fabryki połączeń dostawcy przesyłania komunikatów produktu IBM MQ aplikacja korporacyjna może wyszukać definicję fabryki połączeń i użyć jej w celu utworzenia połączenia 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 następujące wywołanie metody:

```
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, gdy aplikacja korporacyjna jest rozwijana. Ponieważ nazwa JNDI jest zakodowana na stałe w aplikacji, jeśli nazwa JNDI ulegnie zmianie, konieczne jest ponowne zapisanie i ponowne wdrożenie aplikacji.
- Jeśli w ten sposób używana jest definicja fabryki połączeń, nazwa użytkownika i hasło określone w aliasie uwierzytelniania (którego fabryka połączeń została skonfigurowana do użycia) nie są dostępne w dół do IBM MQ. Ma to na celu zapobieganie identyfikowaniu fabryki połączeń przez nieautoryzowane aplikacje oraz możliwość korzystania z niej w celu połączenia z zabezpieczonym systemem IBM MQ .

Nazwa użytkownika i hasło, które są dostępne w produkcie IBM MQ , są zależne od metody, która jest używana do tworzenia połączenia JMS z fabryki połączeń.

Jeśli aplikacja tworzy połączenie produktu JMS przy użyciu metody:

```
ConnectionFactory.createConnection()
```

Domyślna tożsamość użytkownika jest przekazywana do produktu 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 spojrzenie na fabrykę połączeń, a następnie wywołała tę metodę, to nazwa użytkownika i hasło, które zostały przekazane do metody `createConnection()` , są przekazywane do IBM MQ.

Ważne: W wersjach wcześniejszych niż IBM MQ 8.0produkt IBM MQ przetworzył sprawdzenie autoryzacji tylko po to, aby upewnić się, że nazwa użytkownika, który został przyleciony, miała uprawnienia dostępu do menedżera kolejek.

Nie wprowadzono żadnych sprawdzeń w haśle. Aby wykonać sprawdzenie uwierzytelniania i sprawdzić, czy nazwa użytkownika i hasło były poprawne, należy zapisać wyjście zabezpieczeń kanału produktu IBM MQ . Szczegółowe informacje o tym, jak to zrobić, można znaleźć w sekcji Programy obsługi wyjścia zabezpieczeń kanału.

W produkcie IBM MQ 8.0menedżer kolejek sprawdza hasło oprócz nazwy użytkownika.

Korzystanie z fabryki połączeń przy użyciu wyszukiwania pośredniego

Jeśli aplikacja korporacyjna jest pisana, jeśli nazwa JNDI fabryki połączeń jest nieznana lub jeśli aplikacja ma być zainstalowana na różnych serwerach aplikacji przy użyciu innej fabryki połączeń, z inną nazwą produktu JNDI (w zależności od serwera aplikacji, na którym jest zainstalowana), to fabryka połączeń może być wyszukiwaana przy użyciu odwołania do zasobu. Można to zrobić za pomocą wyszukiwania pośredniego.

Przykład

Zamiast bezpośredniego wyszukiwania fabryki połączeń przy użyciu produktu `jms/myCF`aplikacja korporacyjna zawiera odwołanie do zasobu, którego nazwa lokalna produktu JNDI jest następująca: `jms/myResourceReferenceCF`.

Aby użyć tej nazwy JNDI , aplikacja nawiązuje połączenie z repozytorium produktu JNDI serwera aplikacji w taki sam sposób, jak w przypadku, gdy aplikacja wykonuje bezpośrednie wyszukiwanie:

```
InitialContext ctx = new InitialContext();
```

Zamiast bezpośrednio identyfikować produkt `jms/myCF` , aplikacja identyfikuje teraz nazwę JNDI odwołania do zasobu:

```
ConnectionFactory cf = (ConnectionFactory) ctx.lookup("java:comp/env/jms/
myResourceReferenceCF");
```

Należy podać przedrostek `java:comp/env` dla lokalnej nazwy produktu JNDI , aby poinformować serwer aplikacji o tym, że aplikacja korporacyjna wykonuje wyszukiwanie pośrednie.

Po wdrożeniu aplikacji odwzorowuje nazwę JNDI odwołania do zasobu `jms/myResourceReferenceCF` na nazwę JNDI fabryki połączeń, która została już utworzona przez aplikację: `jms/myCF`.

Po uruchomieniu aplikacji wyszukuje on fabrykę połączeń produktu JMS przy użyciu lokalnej nazwy produktu JNDI , na której serwer aplikacji jest odwzorowywać 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, do której odwołuje się zasób, podczas tworzenia połączenia z produktem IBM MQ (jeśli został zdefiniowany alias uwierzytelniania), lub jeśli aplikacja określa własną nazwę użytkownika i hasło.

Wartością domyślną tej właściwości jest *Aplikacja*. Oznacza to, że nazwa użytkownika i hasło, które są dostępne do menedżera kolejek, podczas tworzenia połączenia z produktem JMS , są określane przez samą aplikację. Alias uwierzytelniania fabryki połączeń, do którego nie są używane odwzorowania odwołań do zasobów.

Aplikacje mogą tworzyć połączenia JMS przy użyciu 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 rozwijana w dół do IBM MQ. Jest to nazwa użytkownika i hasło, które uruchomiły serwer aplikacji, na którym działa aplikacja korporacyjna.

Jeśli aplikacja używa produktu `ConnectionFactory.createConnection(String username, String password)`, a opcja **res-auth** jest ustawiona na wartość *Aplikacja*, nazwa użytkownika i hasło przekazywane do metody są wysyłane w dół do IBM MQ.

Aby użyć aliasu uwierzytelniania zdefiniowanego w fabryce połączeń, do której odwołuje się zasób, podczas tworzenia połączenia, należy ustawić właściwość **res-auth** na wartość *Kontener*. Gdy aplikacja tworzy połączenie z serwerem 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 korzysta z odwołania do zasobu z właściwością **res-auth** ustawioną na wartość *Kontener*, można przesłonić alias uwierzytelniania używany podczas tworzenia połączeń produktu 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, do którego aplikacja zostanie wdrożona. Tę właściwość można określić dla wszystkich odwołań do zasobów, które są tworzone przy użyciu narzędzi programu Rational udostępnianych przez produkt IBM.

Przy użyciu tej metody można użyć innego aliasu uwierzytelniania podczas korzystania z fabryki połączeń produktu JMS , która została wyszukana pośrednio. Jeśli podany alias uwierzytelniania nie istnieje, po zainstalowaniu aplikacji korporacyjnej można określić nowy alias. 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 dla komponentów bean sterowanych komunikatami podczas korzystania z klastrów produktu WebSphere Application Server

W przypadku używania aplikacji komponentów bean sterowanych komunikatami wdrożonych w klastrze produktu WebSphere Application Server 7.0 i 8.0 i skonfigurowanych do uruchamiania w trybie normalnym dostawcy przesyłania komunikatów produktu IBM WebSphere MQ , jeden z elementów klastra przetwarza większość komunikatów. Można zrównoważyć obciążenie elementów klastra w celu dystrybucji przetwarzania komunikatów na więcej niż jednym elemencie klastra.

W produkcie IBM WebSphere MQ 7.0 wprowadzono nową funkcję o nazwie **Asynchronous consume**, która umożliwia aplikacjom asynchronicznym korzystanie z komunikatów z kolejki przy użyciu funkcji API o nazwach **MQCB** i **MQCTL**.

Aplikacje komponentu bean sterowanego komunikatami działające w produkcie WebSphere Application Server 7.0 i 8.0, które używają trybu normalnego dostawcy przesyłania komunikatów produktu IBM WebSphere MQ , automatycznie będą korzystać z tej funkcji. Po uruchomieniu aplikacji zostaną one skonfigurowane przez konsumenta asynchronicznego w miejscu docelowym produktu JMS , który został skonfigurowany do monitorowania przez wywołanie funkcji **MQCB**. Interfejs API produktu **MQCTL** jest następnie wywoływany w celu wskazania, że aplikacja jest gotowa do odbierania komunikatów z miejsca docelowego produktu JMS .

Jeśli aplikacje komponentu bean sterowanego komunikatami zostały wdrożone w klastrze produktu WebSphere Application Server , każdy element klastra będzie ustawiał asynchroniczny konsument dla miejsca docelowego produktu JMS , który jest monitorowany przez komponent bean sterowany komunikatami dla komunikatów. Menedżer kolejek produktu IBM WebSphere MQ 7.0 , który udostępnia miejsce docelowe produktu JMS , jest następnie odpowiedzialny za powiadamianie elementu klastra, gdy istnieje odpowiedni komunikat w miejscu docelowym produktu JMS , który ma być przetwarzany.

W wersjach wcześniejszych niż IBM WebSphere MQ 7.0.1 Fix Pack 6menedżery kolejek będą sprzyjać pierwszemu członkowi klastra w celu skonfigurowania asynchronicznego konsumenta w miejscu docelowym JMS. Ten element klastra będzie pierwszym powiadamiany, gdy pojawi się odpowiedni komunikat w miejscu docelowym JMS . Następnie pierwszy element klastra w celu uruchomienia aplikacji komponentu bean sterowanego komunikatami będzie przetwarzał większość odpowiednich komunikatów, które docierają do miejsca docelowego produktu JMS .

Gdy program WebSphere Application Server łączy się z menedżerem kolejek w wersji IBM WebSphere MQ 7.0.1 Fix Pack 6 lub nowszej, komunikaty docierające do miejsca docelowego produktu JMS będą dystrybuowane bardziej równomiernie do wszystkich asynchronicznych konsumentów, które zostały zarejestrowane w tym miejscu docelowym produktu JMS . W przypadku aplikacji komponentów bean sterowanych komunikatami wdrożonych w klastrze serwerów WebSphere Application Server 7.0 i IBM MQ 8.0 oznacza to, że komunikaty będą dystrybuowane bardziej równomiernie między elementami klastra.

Zadania pokrewne

Konfigurowanie właściwości JMS **PROVIDERVERSION**

Korzystanie z pakietu Headers IBM MQ

Pakiet Nagłówki programu IBM MQ udostępnia zestaw pomocniczych interfejsów i klas, których można użyć do manipulowania nagłówkami IBM MQ komunikatu. Zwykle jest używany pakiet IBM MQ Headers, ponieważ usługi administracyjne mają być wykonywane przy użyciu serwera komend (za pomocą komunikatów PCF).

O tym zadaniu

Pakiet Headers IBM MQ znajduje się w pakietach `com.ibm.mq.headers` i `com.ibm.mq.headers.pcf` . Tego narzędzia można użyć zarówno dla dwóch alternatywnych interfejsów API, które program IBM MQ udostępnia do użycia w aplikacjach Java :

- IBM MQ classes for Java (zwany również IBM MQ Base Java).
- IBM MQ classes for Java Message Service (IBM MQ classes for JMS, zwane również IBM MQ JMS).

IBM MQ Base Java applications typically manipulate MQMessage objects, and the Headers support classes can directly interact with these objects, since they natively understand the IBM MQ Base Java interfaces.

W programie IBM MQ JMSładunkiem dla komunikatu jest zazwyczaj łańcuch lub obiekt tablicy bajtów, który może być manipulowany strumieniami DataInput i DataOutput . Pakiet Nagłówki produktu IBM MQ może być używany do interakcji z tymi strumieniami danych i nadaje się do manipulowania dowolnym komunikatami MQ , które są wysyłane i odbierane przez aplikacje produktu IBM MQ JMS .

Oznacza to, że chociaż pakiet Headers produktu IBM MQ zawiera odwołania do pakietu IBM MQ Base Java , jest on również przeznaczony do użytku w aplikacjach produktu IBM MQ JMS i jest odpowiedni do użytku w środowiskach Java Platform, Enterprise Edition (Java EE).

Typowym sposobem, w jaki można użyć pakietu IBM MQ Headers jest manipulowanie komunikatami administracyjnymi w formacie PCF (Programmable Command Format), na przykład z następujących powodów:

- Aby uzyskać dostęp do szczegółowych informacji o zasobie 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 centrycznymi aplikacji można wykonywać z poziomu aplikacji produktu 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 Javamożna znaleźć w sekcji “używanie zIBM MQ classes for Java” na stronie 527.
- Informacje na temat używania pakietu IBM MQ Headers do manipulowania nagłówkami komunikatów dla produktu IBM MQ classes for JMSmożna znaleźć w sekcji “używanie zIBM MQ classes for JMS” na stronie 528.

używanie zIBM MQ classes for Java

Aplikacje produktu IBM MQ classes for Java zwykle manipulują obiektami MQMessage, a klasy obsługi nagłówków mogą bezpośrednio wchodzić w interakcje z tymi obiektami, ponieważ w sposób natywny rozumieją interfejsy produktu IBM MQ classes for Java .

O tym zadaniu

Produkt IBM MQ udostępnia przykładowe aplikacje, które demonstrują sposób korzystania z pakietu Headers produktu IBM MQ przy użyciu interfejsu IBM MQ Base Java API (IBM MQ classes for Java).

Przykłady pokazują dwie rzeczy:

- Jak utworzyć komunikat PCF w celu wykonania czynności administracyjnej i przeanalizowania komunikatu odpowiedzi.
- Sposób wysyłania tego komunikatu PCF za pomocą konsoli IBM MQ classes for Java.

W zależności od używanej platformy, przykłady te są instalowane w katalogu `pcf` w katalogu `samples` lub `tools` instalacji produktu IBM MQ (patrz “Katalogi instalacyjne produktu IBM MQ classes for Java” na stronie 350).

Procedura

1. Utwórz komunikat PCF, aby wykonać działanie administracyjne i przeanalizuj komunikat odpowiedzi.
2. Wyślij ten komunikat PCF za pomocą IBM MQ classes for Java.

Pojęcia pokrewne

“Obsługa nagłówków komunikatów produktu IBM MQ w produkcie IBM MQ classes for Java” na stronie 378
Dostępne są klasy języka Java reprezentujące różne typy nagłówków komunikatów. Dostępne są również dwie klasy pomocnicze.

“Obsługa komunikatów PCF przy użyciu produktu IBM MQ classes for Java” na stronie 383
Klasy Java są udostępniane do tworzenia i analizowania komunikatów ustrukturyzowanych PCF, a także do ułatwiania wysyłania żądań PCF i zbierania odpowiedzi PCF.

używanie zIBM MQ classes for JMS

Aby korzystać z nagłówków IBM MQ z IBM MQ classes for JMS, należy wykonać te same czynności, co w przypadku produktu IBM MQ classes for Java. Komunikat PCF może zostać utworzony, a odpowiedź przeanalizowana dokładnie w ten sam sposób, używając pakietu IBM MQ Headers i tego samego przykładowego kodu, co w przypadku produktu IBM MQ classes for Java.

O tym zadaniu

Aby wysłać komunikat PCF za pomocą interfejsu API produktu IBM MQ , ładunek komunikatu musi zostać zapisany w bajtach produktu JMS i został wysłany przy użyciu standardowych interfejsów API produktu JMS . Jedyna uwaga dotyczy tego, że komunikat nie może zawierać JMS RFH2 ani żadnych innych nagłówków o konkretnych wartościach w strukturze MQMD.

Aby wysłać komunikat PCF, wykonaj następujące kroki. Sposób tworzenia komunikatu PCF, a informacje są wyodrębniane z komunikatu odpowiedzi są takie same, jak w przypadku produktu IBM MQ classes for Java (patrz sekcja “używanie zIBM MQ classes for Java” na stronie 527).

Procedura

1. Utwórz miejsce docelowe kolejki JMS , które reprezentuje SYSTEM.ADMIN.COMMAND.QUEUE.

   Aplikacje programu IBM MQ JMS wysyłają komunikaty PCF do systemu SYSTEM.ADMIN.COMMAND.QUEUEi musi mieć dostęp 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 używany jest produkt WebSphere Application Server, należy zdefiniować te właściwości jako właściwości niestandardowe w miejscu docelowym.

   Aby utworzyć miejsce docelowe w sposób programowy 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 programu JMS B zawierający poprawne wartości MQMD.

   Konieczne jest utworzenie komunikatu JMS B i napisanego do niego komunikatu PCF. Należy utworzyć kolejkę odpowiedzi, ale nie musi ona mieć żadnych konkretnych ustawień.

   W poniższym przykładowym fragmencie kodu przedstawiono sposób tworzenia komunikatu w bajtach produktu JMS i zapisu w nim obiektu com.ibm.mq.headers,pcf.PCFMessage . Obiekt PCFMessage (pcfCmd) został wcześniej zbudowany przy użyciu pakietu Headers IBM MQ . (Należy zwrócić uwagę na pakiet, który ma załadować komunikat PCFMessage: 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 przy użyciu standardowych interfejsów API produktu JMS .
4. Przekształć komunikat odpowiedzi w komunikat PCF w celu przetworzenia.

   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 138

Komunikaty produktu JMS składają się z nagłówka, właściwości i treści. JMS definiuje pięć typów treści komunikatu.

IBM i Setting up IBM MQ on IBM i with Java and JMS

This collection of topics gives an overview of how you set up and test IBM MQ with Java and JMS on IBM i using CL commands or the qshell environment.

Uwaga: Produkty IBM MQ 8.0, `ldap.jar`, `jndi.jar` i `jta.jar` są częścią pakietu JDK.

Korzystanie z komend CL

Ustawiona zmienna CLASSPATH jest używana do testowania z podstawowym językiem Java produktu MQ , usługą JMS z interfejsem JNDI oraz usługą JMS bez interfejsu JNDI.

Jeśli w katalogu `/home/Userprofile` nie zostanie użyty plik `.profile` , konieczne będzie ustawienie poniższych zmiennych środowiskowych na poziomie systemu. Można sprawdzić, czy są 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 danego zadania, wydaj komendę: **WRKENVVAR LEVEL (*JOB)**
3. Jeśli zmienna CLASSPATH nie jest ustawiona, wykonaj następujące czynności:

```
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.mqjms.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 wartość QIBM_MULTI_THREADED nie jest ustawiona, wywołaj następującą komendę:

```
ADDENVVAR ENVVAR(QIBM_MULTI_THREADED) VALUE('Y') LEVEL(*SYS)
```

5. Jeśli wartość QIBM_USE_DESCRIPTOR_STDIO nie jest ustawiona, należy wywołać następującą komendę:

```
ADDENVVAR ENVVAR(QIBM_USE_DESCRIPTOR_STDIO) VALUE('I') LEVEL(*SYS)
```

6. Jeśli parametr QSH_REDIRECTION_TEXTDATA nie jest ustawiony, wywołaj następującą komendę:

```
ADDENVVAR ENVVAR(QSH_REDIRECTION_TEXTDATA) VALUE('Y') LEVEL(*SYS)
```

Korzystanie ze środowiska qshell

Jeśli używane jest środowisko QSHELL, można skonfigurować `.profile` w katalogu `/home/Userprofile` . Więcej informacji na ten temat zawiera dokumentacja programu Qshell Interpreter (qsh).

W polu `.profile`podaj następujące informacje. Należy zauważyć, że instrukcja CLASSPATH musi znajdować się w jednym wierszu lub musi być oddzielona w różnych wierszach za pomocą znaku \ , jak to pokazano na rysunku.

```
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.mqjms.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, wydając komendę **DSPLIBL**.

Jeśli biblioteka QMQMJAVA nie znajduje się 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(tm) 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(tm) 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(tm) Message Service Administration

InitCtx>
InitCtx>
InitCtx>
InitCtx>
InitCtx>
Stopping MQSeries classes for Java(tm) 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(tm) Message Service
Installation Verification Test

Using administered objects, please ensure that these are available

Retrieving a QueueConnectionFactory from JNDI
Creating a Connection
Creating a Session
Retrieving a Queue from JNDI
Creating a QueueSender
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 produktu Java dla produktu IBM MQprzy użyciu repozytorium Maven w celu automatycznego instalowania zależności nie ma potrzeby jawnego instalowania czegokolwiek przed użyciem interfejsów produktu IBM MQ .

Repozytorium centralne narzędzia Maven

Narzędzie Maven jest narzędziem do budowania aplikacji, a także udostępnia repozytorium do przechowywania artefaktów, do których aplikacja może mieć dostęp.

Repozytorium narzędzia Maven (lub repozytorium centralne) umożliwia tworzenie różnych wersji plików, takich jak pliki JAR, które następnie 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 podczas korzystania z narzędzia Maven jako narzędzia budowania jest nazywana plikiem POM, po prostu nazywasz zależności i proces budowania wie, co z niej zrobić.

Pliki klienta IBM MQ

Kopie interfejsów klienta IBM MQ Java są dostępne w centralnym repozytorium pod `com.ibm.mq` `GroupId`. Można znaleźć zarówno `com.ibm.mq.allclient.jar` (zwykle używany dla programów autonomicznych), jak i `wmq.jmsra.rar` (do użytku na serwerach aplikacji Java EE). `allclient.jar` zawiera zarówno IBM MQ classes for JMS , jak i IBM MQ classes for Java.

Ważne: Using the Apache Maven Assembly Plugin *jar-with-dependencies* format to build an application which includes the IBM MQ relocatable JAR file is not supported.

W pliku `pom.xml` przetworzonym przez komendę maven należy dodać zależności dla tych plików JAR, tak jak przedstawiono to w poniższych przykładach:

- Aby wyświetlić relację między kodem aplikacji a `com.ibm.mq.allclient.jar`:

```
<dependency>
        <groupId>com.ibm.mq</groupId>
        <artifactId>com.ibm.mq.allclient</artifactId>
        <version>9.1.3.0</version>
    </dependency>
```

- W przypadku korzystania z adaptera zasobów Java EE :

```
<dependency>
     <groupId>com.ibm.mq</groupId>
     <artifactId>wmq.jmsra</artifactId>
```

```
    <version>9.1.3.0</version>
<dependency/>
```

For an example of a simple project in Eclipse to run a JMS project, see the IBM Developer article Projektowanie aplikacji Java dla MQ po prostu było łatwiejsze z Maven.

Tworzenie aplikacji C++

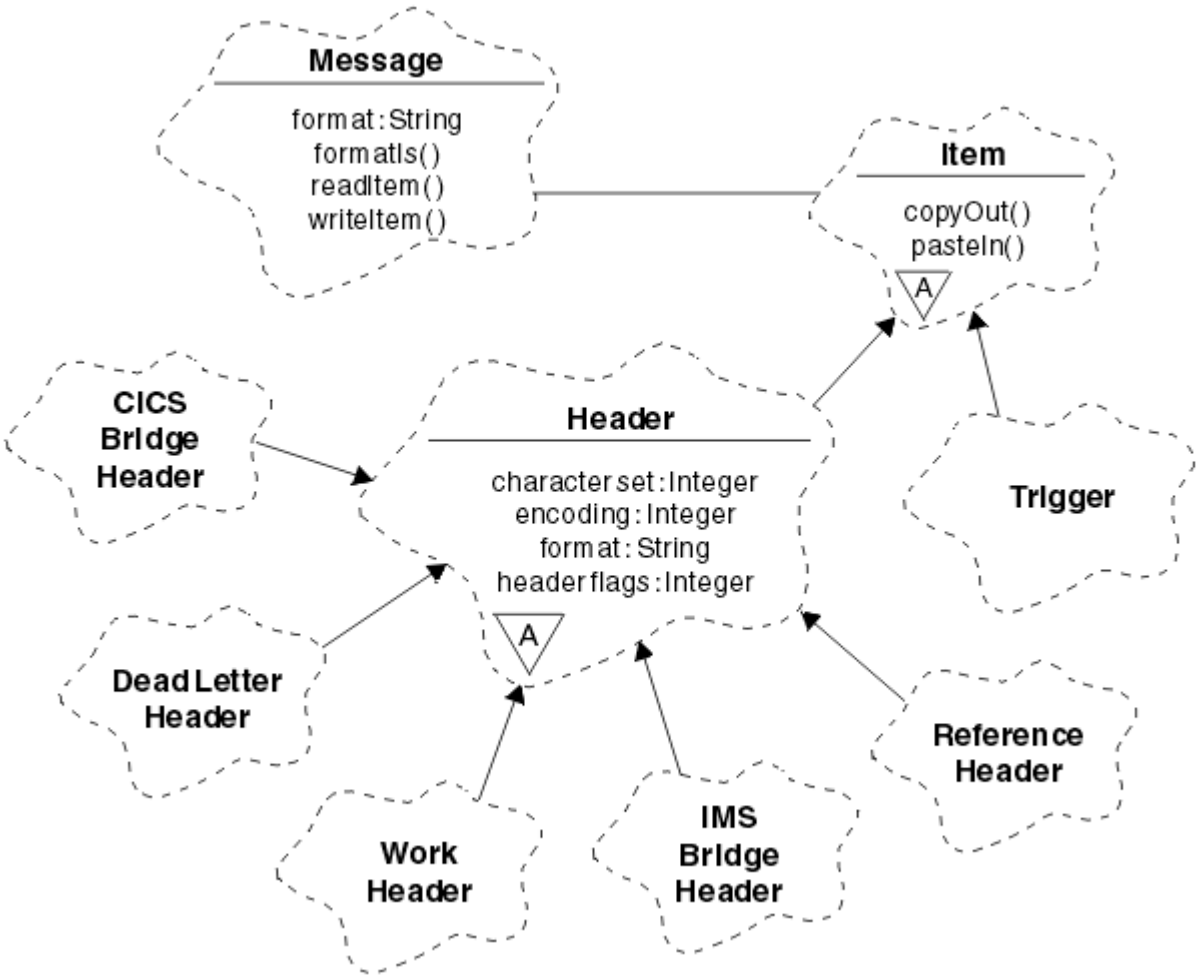
Produkt IBM MQ udostępnia klasy języka C + + równoważne obiektom produktu IBM MQ , a niektóre dodatkowe klasy są równoważne z typami danych tablicowych. Udostępnia ona wiele funkcji, które nie są dostępne w interfejsie MQI.

IBM WebSphere MQ 7.0, rozszerzenia interfejsów programistycznych produktu IBM MQ nie są stosowane w klasach języka C + +.

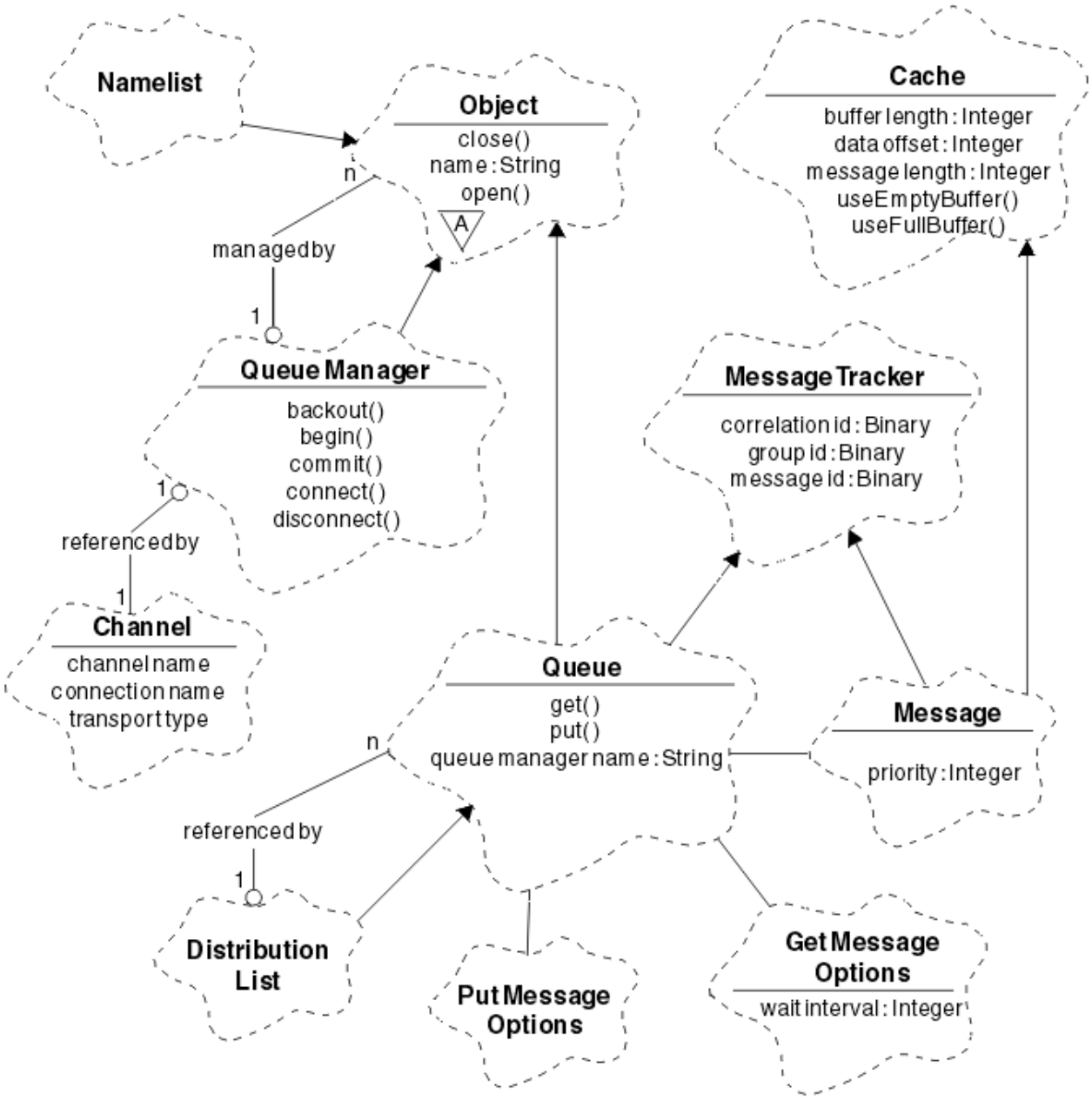
W programie IBM MQ C++ dostępne są następujące funkcje:

- Automatyczne inicjowanie struktur danych produktu IBM MQ .
- Połączenie z menedżerem kolejek i otwieranie kolejki w trybie just-in-time.
- Niejawne zamknięcie kolejki i rozłączenie menedżera kolejek.
- Przesłanie i przyjęcie nagłówka niedostarczanego listu.
- Przesyłanie i odbieranie nagłówka mostu IMS .
- Transmisja i przyjęcie nagłówka komunikatu referencyjnego.
- Wyzwalanie odbioru komunikatu.
- Przesyłanie i odbieranie nagłówka CICS bridge .
- Przesyłanie i odbieranie nagłówka pracy.
- Definicja kanału klienta.

Poniższe diagramy klasy Booch pokazują, że wszystkie klasy są szeroko równoległe do tych obiektów IBM MQ w proceduralnym MQI (na przykład przy użyciu C), które mają albo uchwyty, albo struktury danych. Wszystkie klasy dziedziczą z klasy ImqError (patrz ImqError klasa C++), co pozwala na powiązanie warunku błędu z każdym obiektem.



Rysunek 56. IBM MQ Klasy C++ (obsługa pozycji)



Rysunek 57. IBM MQ Klasy C++ (zarządzanie kolejkami)

Aby poprawnie zinterpretować diagramy klas Booch, należy pamiętać o następujących konwencjach:

- Metody i godne uwagi atrybuty są wyświetlane pod nazwą *klasa* .
- Mały trójkąt w chmurze oznacza *klasę abstrakcyjną*.
- *Dziedziczenie* jest oznaczane strzałką w klasie nadrzędnej.
- Nieprzyozdobiona linia między chmurami oznacza *relację kooperatywną* między klasami.
- Linia urządzona z liczbą oznacza *relację referencyjną* między dwiema klasami. Liczba wskazuje liczbę obiektów, które mogą uczestniczyć w danej relacji w dowolnym momencie.

Następujące klasy i typy danych są używane w sygnaturach metod C + + w klasach zarządzania kolejkami (patrz Rysunek 57 na stronie 539) i klasy obsługi pozycji (patrz Rysunek 56 na stronie 538):

- Klasa ImqBinary (patrz Klasa ImqBinary C++), która hermetyzuje tablice bajtów, takie jak MQBYTE24.
- Typ danych ImqBoolean , który jest zdefiniowany jako **typedef unsigned char ImqBoolean**.
- Klasa ImqString (patrz Klasa ImqString C++), która hermetyzuje tablice znakowe, takie jak MQCHAR64.

Obiekty ze strukturami danych są podsumowane w ramach odpowiednich klas obiektów. Poszczególne pola struktury danych (patrz Skorowidz języka C++ i MQI) są dostępne przy użyciu metod.

Obiekty z uchwytami są dostępne w hierarchii klas ImqObject (patrz Klasa ImqObject C++) i udostępni interfejsy enkapsulowane do interfejsu MQI. Obiekty tych klas wykazują inteligentne zachowanie, które może zredukować liczbę wywołań metod wymaganych w odniesieniu do proceduralnego interfejsu MQI. Na przykład można ustanowić i odrzucić połączenia menedżera kolejek zgodnie z wymaganiami lub otworzyć kolejkę z odpowiednimi opcjami, a następnie zamknąć ją.

Klasa ImqMessage (patrz Klasa ImqMessage C++) hermetyzuje strukturę danych MQMD, a także działa jako punkt wstrzymujący dla danych użytkownika i *elementy* (patrz "Odczytywanie komunikatów w języku C++" na stronie 549) poprzez udostępnienie buforowanych obiektów buforowych. Można podać bufory o stałej długości dla danych użytkownika i wiele razy używać buforu. Ilość danych znajdujących się w buforze może być różna od jednego do następnego. Alternatywnie system może udostępnić bufor o elastycznej długości i zarządzać nim. Zarówno wielkość bufora (kwota dostępna do odbioru wiadomości), jak i ilość rzeczywiście wykorzystana (ilość bajtów do transmisji lub liczba rzeczywiście odebranych bajtów) stają się ważnymi względami.

Pojęcia pokrewne

Przegląd techniczny

"Przykładowe programy w języku C++" na stronie 540
Dostarczane są cztery przykładowe programy demonstrujące pobieranie i umieszczanie komunikatów.

"Uwagi dotyczące języka C++" na stronie 544
Ta kolekcja tematów zawiera szczegóły dotyczące używania języka C++ oraz konwencji, które należy uwzględnić podczas pisania programów aplikacji, które korzystają z interfejsu MQI (Message Queue Interface).

"Przygotowywanie danych komunikatu w języku C++" na stronie 548
Dane komunikatu są przygotowywane w buforze, który może być dostarczony przez system lub aplikację. Istnieją zalety jednej z metod. Podano przykłady użycia buforu.

"Tworzenie aplikacji dla składnika IBM MQ" na stronie 5
Istnieje możliwość tworzenia aplikacji w celu wysyłania i odbierania komunikatów oraz do zarządzania menedżerami kolejek i powiązanymi zasobami. Produkt IBM MQ obsługuje aplikacje napisane w wielu różnych językach i w różnych ramach.

Odsyłacze pokrewne

"Budowanie programów IBM MQ C++" na stronie 555
Zostanie wyświetlony adres URL obsługiwanych kompilatorów wraz z komendami, które mają być używane do kompilowania, łączenia i uruchamiania programów w języku C + + oraz przykładów na platformach IBM MQ .

Skorowidz języka C++ i MQI

Klasy C++ w programie IBM MQ

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 71 na stronie 541.

MQ_INSTALLATION_PATH reprezentuje katalog wysokiego poziomu, w którym jest zainstalowany produkt IBM MQ .

Tabela 71. Położenie programów przykładowych		
Środowisko	**Katalog zawierający źródło**	**Katalog zawierający zbudowane programy**
AIX	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/samp/bin/ia
IBM i	/QIBM/ProdData/mqm/samp/	(patrz uwaga “1” na stronie 541)
Linux	*MQ_INSTALLATION_PATH*/samp	Brak
Solaris	*MQ_INSTALLATION_PATH*/samp	*MQ_INSTALLATION_PATH*/ samp/bin/as
Windows	*MQ_INSTALLATION_PATH*\tools\cplus\ samples	*MQ_INSTALLATION_PATH*\tools\cplus\ przykłady \bin\vn (patrz uwaga “2” na stronie 541)
z/OS	thlqual.SCSQCPPS	

Uwagi:

1. IBM i Programy utworzone przy użyciu kompilatora ILE C++ dla systemu IBM i znajdują się w bibliotece QMQM. Przykładowe pliki znajdują się w katalogu /QIBM/ProdData/mqm/samp.
2. 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 C++ w systemie Windows” na stronie 561.

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)

Te programy w języku C + + umieszczają komunikaty i pobierają komunikaty z kolejki o określonej nazwie.

Te przykłady przedstawiają użycie następujących klas:

- ImqError (patrz sekcja ImqError klasa C++)
- ImqMessage (patrz sekcja ImqMessage C++ class)
- ImqObject (patrz klasa ImqObject C++ class)
- ImqQueue (patrz klasa ImqQueue C++ class)

- Menedżer ImqQueue(patrz sekcja ImqQueue)

Postępuj zgodnie z odpowiednimi instrukcjami, aby uruchomić programy.

Na wszystkich platformach z wyjątkiem z/OS

1. Uruchom komendę **imqsputs** *nazwa-kolejki*.
2. Wpisz linie tekstu w konsoli. Wiersze te są umieszczane jako komunikaty w określonej kolejce.
3. Wprowadź pusty wiersz, aby zakończyć wprowadzanie danych.
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 systemach z/OS Batch, RRS Batch i CICS” na stronie 563.

wł.z/OS

z/OS

1. Skonstruuj i uruchom zadanie wsadowe, używając przykładowego zadania JCL **imqsputr**. Komunikaty są odczytyane z zestawu danych SYSIN.
2. Skonstruuj i uruchom zadanie wsadowe przy użyciu przykładowego skryptu JCL **imqsgetr**. Komunikaty są pobierane z kolejki i wysyłane do zestawu danych SYSPRINT.

Przykładowy program DPUT (imqdput.cpp)

Ten przykładowy program C++ umieszcza komunikaty do listy dystrybucyjnej składającej się z dwóch kolejek.

Funkcja DPUT pokazuje użycie klasy ImqDistributionList (patrz ImqDistributionList C++ class). Ten przykład nie jest obsługiwany w produkcie z/OS.

1. Uruchom komendę **imqdputs** *queue-name-1 queue-name-2* , aby umieścić komunikaty w dwóch nazwanych kolejkach.
2. Uruchom komendę **imqsgets** *queue-name-1* i **imqsgets** *queue-name-2* , aby pobrać komunikaty z tych kolejek.

Uwagi dotyczące języka C++

Ta kolekcja tematów zawiera szczegóły dotyczące używania języka C++ oraz konwencji, które należy uwzględnić podczas pisania programów aplikacji, które korzystają z interfejsu MQI (Message Queue Interface).

Pliki nagłówkowe C++

Pliki nagłówkowe są udostępniane jako część definicji interfejsu MQI w celu ułatwienia pisania programów aplikacji IBM MQ w języku C + +.

Te pliki nagłówkowe są podsumowane w poniższej tabeli.

Tabela 72. Pliki nagłówkowe C/C++

Nazwa pliku	**Spis treści**
IMQI.HPP	Klasy MQI języka C++ (zawiera CMQC.H i IMQTYPE.H)
IMQTYPE.H	Definiuje typ danych **ImqBoolean**
CMQC.H	Struktury danych MQI i stałe manifestów

Aby zwiększyć przenośność aplikacji, należy zakodować nazwę pliku nagłówkowego małymi literami w dyrektywie preprocesora **#include** :

```
#include <imqi.hpp> // C++ classes
```

Metody i atrybuty w języku C++

Nazwy metod znajdują się w mieszanym przypadku. Do parametrów i zwracanych wartości mają zastosowanie różne zagadnienia. Dostęp do atrybutów można uzyskać, używając odpowiednio ustawionych metod i metod pobierania.

Parametry metod, które są *const* , są przeznaczone tylko dla danych wejściowych. Parametry z podpisami łącznie ze wskaźnikiem (*) lub odwołaniem (&) są przekazywane przez referencję. Wartości zwracane, które nie zawierają wskaźnika lub odwołania, są przekazywane przez wartość; w przypadku zwracanych obiektów są to nowe obiekty, które są odpowiedzialne za program wywołujący.

Niektóre sygnatury metod zawierają elementy, które przyjmują wartość domyślną, jeśli nie została określona. Takie elementy są zawsze pod koniec podpisów i są oznaczane znakiem równości (=); wartość po znaku równości wskazuje wartość domyślną, która ma zastosowanie, jeśli element jest pominięty.

Wszystkie nazwy metod w tych klasach są małymi literami, zaczynając od małej litery. Każde słowo, z wyjątkiem pierwszego w nazwie metody, rozpoczyna się od litery. Skróty nie są używane, chyba że ich znaczenie jest szeroko rozumiane. Używane skróty to *id* (dla tożsamości) i *sync* (dla synchronizacji).

Dostęp do atrybutów obiektu można uzyskać za pomocą zestawu i metod pobierania. Metoda set rozpoczyna się od słowa *set* ; Metoda get nie ma przedrostka. Jeśli atrybut ma wartość *tylko do odczytu*, nie ma ustawionej metody.

Atrybuty są inicjowane do poprawnych stanów podczas budowy obiektu, a stan obiektu jest zawsze spójny.

Typy danych w języku C++

Wszystkie typy danych są definiowane przez instrukcję C **typedef** .

Typ **ImqBoolean** jest zdefiniowany jako **unsigned char** w elemencie IMQTYPE.H i może mieć wartości PRAWDA i FAŁSZ. Obiektów klasy **ImqBinary** można używać w miejsce tablic **MQBYTE** , a obiekty klasy **ImqString** w miejsce **char ***. Wiele metod zwraca obiekty, a nie wskaźniki **char** i **MQBYTE** , aby ułatwić zarządzanie pamięcią masową. Wszystkie wartości zwracane stają się odpowiedzialnością programu wywołującego, a w przypadku zwracanego obiektu można usunąć pamięć masową przy użyciu usuwania.

Manipulowanie łańcuchami binarnymi w języku C++

Łańcuchy danych binarnych są deklarowane jako obiekty klasy **ImqBinary** . Obiekty tej klasy mogą być kopiowane, porównywane i ustawiane przy użyciu znanych operatorów języka C. Dostępny jest 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 **znak *** przy użyciu operatora konwersji. Klasa ImqString zawiera metody pomocne przy przetwarzaniu łańcuchów znaków.

Gdy dane znakowe są akceptowane lub zwracane za pomocą metod języka C++ w języku C++, dane znakowe są zawsze zakończone znakiem o kodzie zero i mogą mieć dowolną długość. Jednak niektóre limity są nakładane przez produkt IBM MQ , który może spowodować obcinanie informacji. Aby ułatwić zarządzanie pamięcią masową, dane znakowe są często zwracane w obiektach klasy **ImqString** . Obiekty te można rzutować na **znak *** za pomocą udostępnionego operatora konwersji, a używane w celach *tylko do odczytu* w wielu sytuacjach, w których wymagany jest znak **char *** .

Uwaga: Wynik konwersji **char *** z obiektu klasy **ImqString** może mieć wartość NULL.

Mimo że funkcje języka C mogą być używane na **znaku ***, 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 odzwierciedlający ich atrybuty. Wartości początkowe są zdefiniowane w opisach klas.

Korzystanie z C z C++

Jeśli używane są funkcje C z programu C++, należy dołączyć odpowiednie nagłówki.

W poniższym przykładzie przedstawiono program `string.h` dołączony do programu w języku C + +:

```
extern "C" {
#include <string.h>
}
```

Konwencje notacyjne języka C++

W tym przykładzie przedstawiono sposób wywoływania metod i deklarowania parametrów.

W tym przykładzie kodu używane są metody i parametry **ImqBoolean ImqQueue**:: **get** (**ImqMessage &** *msg*)

Zadeklaruj i wykorzystaj parametry 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++

Kilka operacji może wystąpić w sposób niejawny, *właśnie w czasie*, w celu spełnienia wymagań wstępnych dla pomyślnego wykonania metody. Te niejawne operacje są połączone, otwarte, ponownie otwarte,

zamykane i rozłączane. Za pomocą atrybutów klasy można sterować łączem i otwierać zachowanie niejawne.

Połączenie

Obiekt menedżera ImqQueuejest połączony automatycznie dla dowolnej metody, która powoduje wywołanie interfejsu MQI (patrz Skorowidz języka C++ i MQI).

Otwórz

Obiekt ImqObject jest otwierany automatycznie dla dowolnej metody, której wynikiem jest wywołanie MQGET, MQINQ, MQPUT lub MQSET. Użyj metody **openFor** , aby określić jedną lub większą liczbę odpowiednich wartości **open option** .

Otwórz ponownie

Obiekt ImqObject jest ponownie otwierany automatycznie dla dowolnej metody, której wynikiem jest wywołanie MQGET, MQINQ, MQPUT lub MQSET, w którym obiekt jest już otwarty, ale istniejące **opcje otwarcia** nie są odpowiednie, aby umożliwić pomyślne wywołanie MQI. Obiekt jest tymczasowo zamknięty przy użyciu tymczasowej wartości **close options** parametru MQCO_NONE. Użyj metody **openFor** , aby dodać odpowiedni**open option**.

Ponowne otwarcie może powodować problemy w określonych okolicznościach:

- Tymczasowa kolejka dynamiczna jest niszczona, gdy jest zamknięta i nigdy nie może zostać ponownie otwarta.
- Kolejka otwarta dla wejścia na wyłączność (jawnie lub domyślnie) może być dostępna dla innych osób w oknie potencjalnej transakcji podczas zamykania i ponownego otwierania.
- Pozycja kursora przeglądania jest tracona, gdy kolejka jest zamknięta. Ta sytuacja nie zapobiega zamknięciu i ponownym otwieraniu, ale uniemożliwia późniejsze użycie kursora, dopóki nie zostanie ponownie użyta wartość MQGMO_BROWSE_FIRST.
- Kontekst ostatniego pobranego komunikatu jest tracony, gdy kolejka jest zamknięta.

Jeśli którakolwiek z tych okoliczności wystąpi lub może być przewidziana, należy unikać ponownego otwierania, jawnie ustawiając odpowiednie **opcje otwarte** przed otwartymi obiektami (jawnie lub niejawnie).

Ustawienie opcji **open options** w sposób jawny dla złożonych sytuacji związanych z obsługą kolejek powoduje lepszą wydajność i pozwala uniknąć problemów związanych z korzystaniem z ponownie otwartych.

Zamknij

Obiekt ImqObject jest zamykany automatycznie w każdym punkcie, w którym stan obiektu nie jest już wykonalny, na przykład wtedy, gdy odniesienie do połączenia ImqObject zostanie zerwane lub jeśli obiekt ImqObject zostanie zniszczony.

Rozłącz

Menedżer ImqQueuejest odłączony automatycznie w dowolnym momencie, w którym połączenie nie jest już wykonalne, na przykład wtedy, gdy odniesienie do połączenia ImqObject zostanie zerwane lub jeśli obiekt ImqQueueManager zostanie zniszczony.

Łańcuchy binarne i łańcuchy znaków w języku C++

Klasa ImqString hermetyzuje tradycyjny format danych *char** . Klasa ImqBinary hermetyzuje binarną tablicę bajtów. Niektóre metody, które ustawiają dane znakowe, mogą obcinać dane.

Metody, które ustawiają znak (**char ***) Dane zawsze przyjmują kopię danych, ale niektóre metody mogą obciąć kopię, ponieważ niektóre limity są nakładane przez produkt IBM MQ.

Klasa ImqString (patrz Klasa ImqString C++) hermetyzuje tradycyjny produkt **znak *** i zapewnia obsługę:

- Porównanie
- Konkatenacja
- Kopiowanie
- Konwersja typu integer-to-text i text-to-integer
- Wyodrębnianie znacznika (słowa)
- Konwersja wielkich liter

Klasa ImqBinary (patrz Klasa ImqBinary C++) hermetyzuje binarne tablice bajtowe o dowolnej wielkości. W szczególności jest on używany do przechowywania następujących atrybutów:

- **token rozliczania** (MQBYTE32)
- **znacznik połączenia** (MQBYTE128)
- **identyfikator korelacji** (MQBYTE24)
- **facility token (znacznik obiektu)** (MQBYTE8)
- **identyfikator grupy** (MQBYTE24)
- **identyfikator instancji** (MQBYTE24)
- **ID komunikatu** (MQBYTE24)
- **token komunikatu** (MQBYTE16)
- **identyfikator instancji transakcji** (MQBYTE16)

Miejsce, w którym te atrybuty należą do obiektów następujących klas:

- ImqCICSBridgeHeader (patrz Klasa ImqCICSBridgeHeader C++)
- ImqGetMessageOptions (patrz ImqGetMessageOptions C++ class)
- ImqIMSBridgeHeader (patrz Klasa ImqIMSBridgeHeader C++)
- ImqMessageTracker (patrz ImqMessageTracker C++ class)
- Menedżer ImqQueue(patrz sekcja ImqQueue)
- Nagłówek ImqReference(patrz sekcja ImqReferenceHeader C++ class)
- ImqWorkNagłówek (patrz ImqWorkKlasa nagłówka C++)

Klasa ImqBinary udostępnia również obsługę porównywania i kopiowania.

Nieobsługiwane funkcje w języku C++

Klasy i metody języka C++ programu IBM MQ są niezależne od platformy IBM MQ . Mogą one zatem oferować niektóre funkcje, które nie są obsługiwane na określonych platformach.

W przypadku próby użycia funkcji na platformie, na której nie jest ona obsługiwana, funkcja jest wykrywana przez program IBM MQ , ale nie przez powiązania języka C + +. Program IBM MQ zgłasza błąd do programu, podobnie jak inne błędy 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 dostarczony przez system lub aplikację. Istnieją zalety jednej z metod. Podano przykłady użycia buforu.

Po wysłaniu komunikatu dane komunikatu są najpierw przygotowywane w buforze zarządzanym przez obiekt ImqCache (patrz sekcja ImqCache). Bufor jest powiązany (przez dziedziczenie) z każdym obiektem ImqMessage (patrz klasa ImqMessage C++ class): może być dostarczony przez aplikację

(przy użyciu metody **useEmptyBuffer** lub **useFullBuffer**) lub automatycznie przez system. Zaletą aplikacji dostarczających bufor komunikatów jest to, że kopiowanie danych nie jest konieczne w wielu przypadkach, ponieważ aplikacja może korzystać bezpośrednio z przygotowanych obszarów danych. Wadą jest to, że podany bufor ma stałą długość.

Bufor może być ponownie wykorzystany, a liczba przesłanych bajtów może być zmieniana za każdym razem za pomocą metody **setMessageLength** przed przesłaniem.

Jeśli system jest dostarczany automatycznie przez system, liczba dostępnych bajtów jest zarządzana przez system, a dane mogą być kopiowane do buforu komunikatów za pomocą, na przykład, metody ImqCache **zapis** lub metody ImqMessage **writeItem** . Bufor komunikatów rośnie zgodnie z potrzebą. W miarę wzrostu bufora, nie dochodzi do utraty wcześniej zapisanych danych. Duży lub wieloczęściowy komunikat może być zapisany w kolejnych fragmentach.

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 dostarczanego przez 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 i treść)

```
ImqString strText( "Hello world" );

msg.writeItem( strText );
```

Odczytywanie komunikatów w języku C++

Bufor może być dostarczony przez aplikację lub system. Dostęp do danych możliwy jest bezpośrednio z buforu lub odczytany sekwencyjnie. Dla każdego typu komunikatu istnieje równorzędna klasa. Podano przykładowy kod.

W przypadku odbierania danych aplikacja lub system może dostarczyć odpowiedni bufor komunikatów. Ten sam bufor może być używany zarówno dla wielu transmisji, jak i wielokrotnych przyjęć dla konkretnego obiektu ImqMessage . Jeśli bufor komunikatów jest dostarczany automatycznie, rośnie w celu dostosowania dowolnej długości danych. Jednak bufor komunikatów dostarczony przez aplikację

może nie być wystarczająco duży, aby pomieścić otrzymane dane. Następnie może wystąpić obcięcie lub niepowodzenie, w zależności od opcji użytych do przyjęcia komunikatu.

Dane przychodzące mogą być dostępne bezpośrednio z buforu komunikatów, w którym to przypadku długość danych wskazuje łączną ilość danych przychodzących. Alternatywnie, dane przychodzące mogą być odczytywaniekolejno z buforu komunikatów. W tym przypadku wskaźnik danych zajmuje się następnym bajtem danych przychodzących, a wskaźnik danych i długość danych są aktualizowane za każdym razem, gdy dane są odczytywaane.

Elementy są elementami komunikatu, a wszystko to w obszarze użytkownika buforu komunikatów, które muszą być przetwarzane sekwencyjnie i osobno. Oprócz zwykłych danych użytkownika element może być nagłówkiem niewysłanych wiadomości lub komunikatem wyzwalacza. Elementy są zawsze powiązane z formatami komunikatów. Formaty komunikatów **nie** są zawsze powiązane z elementami.

Istnieje klasa obiektu dla każdej pozycji, która odpowiada rozpoznawalnego formatu komunikatu produktu IBM MQ . Istnieje jeden dla nagłówka niedostarczonych komunikatów i jeden dla komunikatu wyzwalacza. Brak klasy obiektu dla danych użytkownika. Oznacza to, że po wyczerpaniu rozpoznawalnych formatów przetwarzanie pozostałej części jest pozostawione do programu aplikacji. Klasy dla danych użytkownika mogą być zapisywane przez specjalizującą się klasę ImqItem .

W poniższym przykładzie przedstawiono przyjęcie komunikatu, które uwzględnia liczbę potencjalnych elementów, które mogą poprzedzać dane użytkownika w wyimaginowanej sytuacji. Dane użytkownika innego niż element są definiowane jako elementy, które pojawiają się po elementach, które można zidentyfikować. Bufor automatyczny (wartość domyślna) 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 `UserClass`i jest definiowana przez aplikację.

Produkt `UserClass` pochodzi z klasy ImqItem (patrz klasa ImqItem C++ class) i implementuje z tej klasy metody wirtualnego **copyOut** i **pasteIn** .

Kolejne dwa przykłady pokazują kod z klasy ImqDeadLetterHeader (patrz ImqDeadLetterHeader C++ class). W pierwszym przykładzie przedstawiono kod niestandardowy- *zapis* w sposób niestandardowy.

```
// 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- *odczytywanie* .

```
// 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 buforu automatycznego pamięć masowa buforu jest *ulotna*. Oznacza to, że dane buforu mogą być przechowywane w innym miejscu fizycznym po każdym wywołaniu metody **get** . Z tego powodu do każdej przywoływanej danych buforu należy użyć metod **bufferPointer** lub **dataPointer** w celu uzyskania dostępu do danych komunikatu.

Użytkownik może chcieć, aby program zarezerwował stały obszar na potrzeby odbierania danych komunikatu. W tym przypadku należy wywołać metodę **useEmptyBuffer** przed użyciem metody **get** .

Użycie stałego, nieautomatycznego obszaru ogranicza liczbę komunikatów do maksymalnej wielkości, dlatego ważne jest, aby rozważyć opcję MQGMO_ACCEPT_TRUNCATED_MSG obiektu ImqGetMessageOptions . Jeśli ta opcja nie zostanie podana (wartość domyślna), kod przyczyny MQRC_TRUNCATED_MSG_FAILED może być oczekiwany. Jeśli ta opcja jest określona, kod przyczyny MQRC_TRUNCATED_MSG_ACCEPTED może być oczekiwany w zależności od projektu aplikacji.

W następnym przykładzie pokazano, w jaki sposób można wykorzystać stałą przestrzeń pamięci 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, przy użyciu metody *pszBuffer*, a nie za pomocą metody **bufferPointer** . Jednak lepszym sposobem jest użycie metody **dataPointer** w celu uzyskania dostępu ogólnego. Aplikacja (nie obiekt klasy ImqCache) musi odrzucić bufor zdefiniowany przez użytkownika (nieautomatyczny).

Uwaga: Określenie pustego wskaźnika i zerowej długości z parametrem **useEmptyBuffer** nie mianuje buforu o stałej długości równej zero, co może być oczekiwane. Ta kombinacja jest interpretowana jako żądanie do zignorowania wszystkich poprzednich buforów zdefiniowanych przez użytkownika, a zamiast tego jest przywracana do użycia buforu automatycznego.

Zapisywanie komunikatu do kolejki niedostarczonych komunikatów w języku C++

Przykładowy kod programu do zapisu komunikatu do kolejki niedostarczonych komunikatów.

Typowym przypadkiem komunikatu wieloczęściowego jest jeden zawierający nagłówek niedostarczonych komunikatów. Dane z komunikatu, którego nie można przetworzyć, są dopisane do nagłówka niedostarczonych komunikatów.

```
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 zapisu komunikatu do mostu IMS .

Komunikaty wysłane do mostu IBM MQ - IMS mogą używać specjalnego nagłówka. Nagłówek mostu IMS jest poprzedzony przedrostkiem 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 w produkcie CICS bridge w języku C++

Przykładowy kod programu do zapisu komunikatu do CICS bridge.

Komunikaty wysyłane do programu IBM MQ for z/OS przy użyciu CICS bridge wymagają specjalnego nagłówka. Nagłówek CICS bridge jest poprzedzony przedrostkiem 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 );
```

Zapisywanie komunikatu z nagłówkiem pracy w języku C++

Przykładowy kod programu do zapisu komunikatu przeznaczonego dla kolejki zarządzanej przez program z/OS Workload Manager.

Komunikaty wysyłane do programu IBM MQ for z/OS, które są przeznaczone dla kolejki zarządzanej przez menedżera obciążenia produktu z/OS , wymagają specjalnego nagłówka. Nagłówek pracy jest poprzedzony przedrostkiem 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 IBM MQ C++

Zostanie wyświetlony adres URL obsługiwanych kompilatorów wraz z komendami, które mają być używane do kompilowania, łączenia i uruchamiania programów w języku C + + oraz przykładów na platformach IBM MQ .

Listę kompilatorów dla każdej obsługiwanej platformy i wersji produktu IBM MQmożna znaleźć w sekcji Wymagania systemowe produktu IBM MQ.

Komenda, którą należy skompilować i połączyć z programem IBM MQ C++, zależy od instalacji i wymagań. W poniższych przykładach przedstawiono typowe komendy kompilowania i łączenia dla niektórych kompilatorów, które korzystają z domyślnej instalacji produktu IBM MQ na wielu platformach.

AIX Budowanie programów C++ w systemie AIX

Kompilowanie programów IBM MQ C + + w systemie AIX przy użyciu kompilatora XL C Enterprise Edition .

Klient

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

32-bitowa aplikacja wielowątkowa

```
xlC -o imqsputc_32 imqsput.cpp -qchars=signed -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 -qchars=signed -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqc23ia_r -limqb23ia_r -lmqic_r
```

Aplikacja wielowątkowa z 64-bitową wersją

```
xlC -q64 -o imqsputc_64 imqsput.cpp -qchars=signed -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ia -limqb23ia -lmqic
```

Aplikacja wielowątkowa z 64-bitowym

```
xlC_r -q64 -o imqsputc_64_r imqsput.cpp -qchars=signed -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqc23ia_r -limqb23ia_r -lmqic_r
```

Serwer

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

32-bitowa aplikacja wielowątkowa

```
xlC -o imqsput_32 imqsput.cpp -qchars=signed -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 -qchars=signed -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -limqs23ia_r -limqb23ia_r -lmqm_r
```

Aplikacja wielowątkowa z 64-bitową wersją

```
xlC -q64 -o imqsput_64 imqsput.cpp -qchars=signed -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ia -limqb23ia -lmqm
```

Aplikacja wielowątkowa z 64-bitowym

```
xlC_r -q64 -o imqsput_64_r imqsput.cpp -qchars=signed -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -limqs23ia_r -limqb23ia_r -lmqm_r
```

IBM i Budowanie programów C++ w systemie IBM i

Kompilowanie programów IBM MQ C++ na serwerze IBM i przy użyciu kompilatora ILE C++.

IBM ILE C++ for IBM i to rodzimy kompilator dla programów w języku C++. Poniższe instrukcje opisują sposób użycia tego kompilatora do tworzenia aplikacji w języku C + + IBM MQ przy użyciu *Hello World!* Przykładowy program IBM MQ jako przykład.

1. Zainstaluj kompilator ILE C++ for IBM i w sposób opisany 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 C++ można znaleźć w programie `/QIBM/ProdData/mqm/samp` i w plikach włączanych w programie `/QIBM/ProdData/mqm/inc`.

   Alternatywnie, źródło może zostać znalezione w bibliotece `SRCFILE(QCPPSRC/LIB) SRCMBR(IMQWRLD)`.

b. Powiąż to z programami usługowym dostarczonym z produktem IBM MQ, aby utworzyć obiekt programu:

```
CRTPGM PGM(MYLIB/IMQWRLD) MODULE(MYLIB/IMQWRLD) +
BNDSRVPGM(QMQM/IMQB23I4 QMQM/IMQS23I4)
```

Aby zbudować aplikację wielowątkową, należy użyć programów usługowych dla ponownego wejścia:

```
CRTPGM PGM(MYLIB/IMQWRLD) MODULE(MYLIB/IMQWRLD) +
BNDSRVPGM(QMQM/IMQB23I4[_R] QMQM/IMQS23I4[_R])
```

c. Wykonaj przykładowy program HELLO WORLD, używając SYSTEM.DEFAULT.LOCAL.QUEUE:

```
CALL PGM(MYLIB/IMQWRLD)
```

Linux Budowanie programów C++ w systemie Linux

Kompilacja programów IBM MQ C++ na serwerze Linux przy użyciu kompilatora GNU g + +.

System p

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Klient:System p

32-bitowa aplikacja wielowątkowa

```
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
```

Aplikacja wielowątkowa z 64-bitową wersją

```
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 wielowątkowa z 64-bitowym

```
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 wielowątkowa

```
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
```

Aplikacja wielowątkowa z 64-bitową wersją

```
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 wielowątkowa z 64-bitowym

```
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 najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Klient:IBM Z

32-bitowa aplikacja wielowątkowa

```
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
```

Aplikacja wielowątkowa z 64-bitową wersją

```
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 wielowątkowa z 64-bitowym

```
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 wielowątkowa

```
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
```

Aplikacja wielowątkowa z 64-bitową wersją

```
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 wielowątkowa z 64-bitowym

```
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 najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Klient: x86-64 (32-bitowy)

32-bitowa aplikacja wielowątkowa

```
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
```

Aplikacja wielowątkowa z 64-bitową wersją

```
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 wielowątkowa z 64-bitowym

```
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 wielowątkowa

```
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
```

Aplikacja wielowątkowa z 64-bitową wersją

```
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 wielowątkowa z 64-bitowym

```
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
```

Solaris Budowanie programów C++ w systemie Solaris

Kompilowanie programów IBM MQ C++ na serwerze Solaris przy użyciu kompilatora Sun ONE.

SPARC

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Klient: SPARC

Aplikacja 32-bitowa

```
CC -xarch=v8plus -mt -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqc23as -limqb23as
-lmqic -lsocket -lnsl -ldl
```

Aplikacja 64-bitowa

```
CC -xarch=v9 -mt -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqc23as
-limqb23as
-lmqic -lsocket -lnsl -ldl
```

Serwer: SPARC

Aplikacja 32-bitowa

```
CC -xarch=v8plus -mt -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqs23as -limqb23as
-lmqm -lsocket -lnsl -ldl
```

Aplikacja 64-bitowa

```
CC -xarch=v9 -mt -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
```

```
-limqb23as
-lmqm -lsocket -lnsl -ldl
```

x86-64

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Klient: x86-64

Aplikacja 32-bitowa

```
CC -xarch=386 -mt -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqc23as -limqb23as
-lmqic -lsocket -lnsl -ldl
```

Aplikacja 64-bitowa

```
CC -xarch=amd64 -mt -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqc23as
-limqb23as
-lmqic -lsocket -lnsl -ldl
```

Serwer: x86-64

Aplikacja 32-bitowa

```
CC -xarch=386 -mt -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqs23as -limqb23as
-lmqm -lsocket -lnsl -ldl
```

Aplikacja 64-bitowa

```
CC -xarch=amd64 -mt -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
-limqb23as
-lmqm -lsocket -lnsl -ldl
```

Windows Budowanie programów C++ w systemie Windows

Kompilowanie programów IBM MQ C++ na serwerze Windows przy użyciu kompilatora Microsoft Visual Studio C++.

Ostrzeżenie: Biblioteki dostarczane przez produkt IBM MQ są bibliotekami dynamicznymi, a nie bibliotekami statycznymi. Produkt IBM MQ udostępnia nazwę "`import libraries`", która może być używana tylko w czasie kompilacji. W przypadku środowiska wykonawczego należy używać bibliotek dynamicznych.

Z poziomu produktu IBM MQ 8.0.0 Fix Pack 4produkt IBM MQ jest dostarczany z redystrybuowanymi klientami, które zawierają biblioteki wymagane do uruchamiania aplikacji produktu IBM MQ . Te biblioteki mogą być spakowane i redystrybuowane za pomocą aplikacji klienckich. Więcej informacji na ten temat zawiera sekcja Redistributable clients on Windows(klienty podlegające redystrybucji w systemie

Pliki biblioteki (.lib) i pliki dll do użytku z aplikacjami 32-bitowymi są instalowane w programie *MQ_INSTALLATION_PATH*`/Tools/Lib`. Pliki przeznaczone do użytku z aplikacjami 64-bitowymi są instalowane w programie *MQ_INSTALLATION_PATH*`/Tools/Lib64`. *MQ_INSTALLATION_PATH* reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest 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 serwer Windows 8.1 lub Windows Server 2012 R2, należy zainstalować uniwersalną aktualizację środowiska wykonawczego C (Universal C runtime update-Universal CRT) z produktu Microsoft. To środowisko wykonawcze jest częścią produktów Windows 10 i Windows Server 2016.

Uniwersalna aktualizacja CRT to Microsoft update KB3118401. W celu sprawdzenia, czy ta aktualizacja jest dostępna, można 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, który został skompilowany samodzielnie za pomocą programu Microsoft Visual Studio 2017, bez zainstalowanego środowiska wykonawczego powoduje 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.
```

Tworzenie 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 produktu IBM MQ nie instaluje środowiska wykonawczego programu Microsoft Visual Studio 2012 C/C + +. Jeśli poprzednia wersja produktu IBM MQ została zainstalowana na tym samym komputerze, to środowiska wykonawcze Microsoft Visual Studio 2012 są dostępne w tej instalacji.

Jeśli jednak używany jest program, który został zbudowany przy użyciu produktu Microsoft Visual Studio 2012 i nie została zainstalowana żadna wcześniejsza wersja produktu IBM MQ , należy wykonać jedną z następujących czynności:

- Pobierz i zainstaluj **Microsoft Visual C++ Redistributable for VisualStudio 2017 (32 and 64-bit versions)** z programu Microsoft.
- Ponownie skompiluj program z programem Microsoft Visual Studio 2017lub innym poziomem Microsoft Visual Studio , dla którego zainstalowane są środowiska wykonawcze.

Biblioteki klienta C++ zbudowane za pomocą kompilatora Microsoft Visual Studio 2015

Produkt IBM MQ udostępnia biblioteki klienta C + +, które są budowane przy użyciu kompilatora Microsoft Visual Studio 2015 C++ oraz kompilator języka C++ w wersji Microsoft Visual Studio 2017 .

Dostępne są zarówno 32-bitowe, jak i 64-bitowe wersje bibliotek C++ programu IBM MQ . Biblioteki 32-bitowe są instalowane w folderze `bin\vs2015` , a biblioteki 64-bitowe są instalowane w folderach `bin64\vs2015` .

Domyślnie produkt IBM MQ jest skonfigurowany pod kątem korzystania z bibliotek produktu Microsoft Visual Studio 2017 . Aby użyć bibliotek produktu 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 różnie nazwanych bibliotek C++ IBM MQ

Produkt IBM MQ udostępnia kilka dodatkowych bibliotek klienta C + +, które zostały nazwane inaczej. Biblioteki te są budowane przy użyciu kompilatorów Microsoft Visual Studio 2015 i Microsoft Visual Studio 2017 C + +. Biblioteki te 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 produktu IBM MQ C + + mają różne nazwy, można uruchamiać aplikacje IBM MQ C + +, które są budowane przy użyciu programu IBM MQ C++ i skompilowane z produktem Microsoft Visual Studio 2017 i wcześniejszymi wersjami produktu na tym samym komputerze.

Dodatkowe biblioteki produktu Microsoft Visual Studio 2017 mają następujące nazwy:

- `imqb23vnvs2017.dll`
- `imqc23vnvs2017.dll`
- `imqs23vnvs2017.dll`
- `imqx23vnvs2017.dll`

Dodatkowe biblioteki produktu 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 skompilować ją za pomocą kompilatora Microsoft Visual Studio 2015 . Aby uruchomić aplikacje w języku C++ IBM MQ skompilowane z produktem Microsoft Visual Studio 2015lub aplikacje skompilowane z wcześniejszą wersją produktu na tym samym komputerze, zmienna środowiskowa PATH musi być poprzedzona przedrostkiem, tak jak pokazano to 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%
```

Odsyłacze pokrewne

Windows: zmiany z IBM MQ 8.0

z/OS Budowanie programów w języku C + + w systemach z/OS Batch, RRS Batch i CICS

Tworzenie programów IBM MQ C++ na serwerze z/OS dla środowisk wsadowych, środowisk wsadowych RRS lub CICS oraz uruchamianie programów przykładowych.

Programy w języku C + + można pisać dla trzech środowisk obsługiwanych przez produkt IBM MQ for z/OS :

- Wsadowe
- Zadanie wsadowe RRS
- CICS

Skompiluj, prelink i link

Tworzenie aplikacji produktu z/OS przez kompilowanie, wstępne połączenie i tworzenie odsyłaczy do edycji kodu źródłowego C++.

IBM MQ C++ for z/OS is implemented as z/OS DLLs for the IBM C++ for z/OS language. Za pomocą bibliotek DLL można konkatenować dostarczonej definicji sidedecks z danymi wyjściowymi kompilatora w czasie przed połączeniem. Dzięki temu konsolidator może sprawdzić połączenia z funkcjami składową języka C++ w programie IBM MQ .

Uwaga: Dla każdego z trzech środowisk dostępne są trzy zestawy sidedecks.

Aby zbudować aplikację w języku C++ w wersji IBM MQ for z/OS , utwórz i uruchom JCL. Użyj następującej procedury:

1. Jeśli aplikacja jest uruchamiana w produkcie CICS, należy użyć procedury CICS, aby przetłumaczyć komendy produktu CICS w programie.

   Ponadto w przypadku aplikacji produktu CICS należy wykonać następujące czynności:

   a. Dodaj bibliotekę SCSQLOAD do konkatenacji DFHRPL.

   b. Zdefiniuj grupę CEDA CSQCAT1 , korzystając z elementu IMQ4B100 w bibliotece SCSQPROC.

   c. Zainstaluj CSQCAT1.
2. Skompiluj program, aby utworzyć kod obiektu. Kod JCL dla kompilacji musi zawierać instrukcje, które sprawiają, że pliki definicji danych produktu są dostępne dla kompilatora. Definicje danych są dostarczane w następujących bibliotekach produktu IBM MQ for z/OS :

   - **thlqual**.SCSQC370
   - **thlqual**.SCSQHPPS

   Upewnij się, że podano opcję kompilatora `/cxx` .

   Uwaga: Nazwa **thlqual** jest kwalifikatorem wysokiego poziomu dla biblioteki instalacji produktu IBM MQ w systemie z/OS.
3. Utwórz przed dowiązaniem kod obiektu utworzony w kroku “2” na stronie 564, w tym następujące definicje sidedecks, 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 dla 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 CICS
4. Odsyłacz-edytuj kod obiektu utworzony w kroku “3” na stronie 564, aby utworzyć moduł ładowalny i zapisać go w bibliotece ładowania aplikacji.

Aby uruchomić wsadowy lub RRS programy wsadowe, należy dołączyć biblioteki **thlqual**.SCSQAUTH i **thlqual**.SCSQLOAD w konkatenacji zestawu danych STEPLIB lub JOBLIB.

Aby uruchomić program CICS , najpierw należy skontaktować się z administratorem systemu, aby zdefiniować go jako CICS jako program i transakcję IBM MQ . Następnie można go uruchomić w zwykły sposób.

Uruchamianie przykładowych programów

Programy są opisane w sekcji “Przykładowe programy w języku C++” na stronie 540.

Aplikacje przykładowe są dostarczane tylko w formularzu źródłowym. Pliki są następujące:

Tabela 73. Przykładowe pliki programu z/OS		
Przykład	**Program źródłowy (w bibliotece thlqual.SCSQCPPS)**	**JCL (w bibliotece thlqual.SCSQPROC)**
Hello World	imqwrld	imqwrldr
SPUT	imqsput	imqsputr
SGET	imqsget	imqsgetr

Aby uruchomić przykłady, należy je skompilować i utworzyć odsyłacz-edytuj je tak, jak w przypadku dowolnego programu C++ (patrz “Budowanie programów w języku C + + w systemach z/OS Batch, RRS Batch i CICS” na stronie 563). Użyj dostarczonego kodu JCL do utworzenia i uruchomienia zadania wsadowego. Najpierw należy dostosować JCL, postępując zgodnie z komentarzem dołączonym do niego.

z/OS Budowanie programów C++ w usługach systemowych z/OS UNIX

Tworzenie programów w języku C++ w IBM MQ C++ w systemie z/OS dla usług systemu Unix.

Aby zbudować aplikację w powłoce UNIX System Services, należy nadać kompilatorowi dostęp do plików włączanych w wersji IBM MQ (znajdujących się w katalogu `thlqual.SCSQC370` i `hlqual.SCSQHPPS`), a także połączyć się z dwoma bibliotekami DLL sidedecks (znajdującymi się w katalogu `thlqual.SCSQDEFS`). W czasie wykonywania aplikacja wymaga dostępu do zestawów danych IBM MQ `thlqual.SCSQLOAD`, `thlqual.SCSQAUTH`oraz jednego z zestawów danych specyficznych dla języka, takich jak `thlqual.SCSQANLE` .[6].

Kompilacja

1. Skopiuj próbkę do systemu plików za pomocą komendy TSO **oput** lub użyj protokołu FTP. W dalszej części tego przykładu założono, że próbka została skopiowana do katalogu o nazwie `/u/fred/samplei` nazwano go `imqwrld.cpp`.
2. Zaloguj się do powłoki usług systemowych produktu UNIX i przejdź do katalogu, w którym została umieszczona próba.
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"
```

4. Skompiluj i dowiązaj przykładowy program. Poniższa komenda łączy program z sydedecks zadania wsadowego. Zamiast tego można użyć sydedecks zadania wsadowego RRS. Znak \ jest używany do rozdzielenia komendy na więcej niż jedną linię. Nie wprowadzaj tego znaku; wprowadź komendę jako jeden wiersz:

```
/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** można znaleźć w podręczniku z/OS UNIX System Services Command Reference.

Można również użyć programu narzędziowego make, aby uprościć tworzenie programów w języku C + +. Poniżej znajduje się przykładowy plik makefile do zbudowania przykładowego programu HELLO WORLD

[6] Istnieje możliwość powiązania z dowolnymi z wymienionych w sekcji " Wstępnie dowiązaniem kodu obiektu w celu uruchomienia usługi systemowej UNIX w dowolnym z trzech środowisk, “Budowanie programów w języku C + + w systemach z/OS Batch, RRS Batch i CICS” na stronie 563

+ +. Separuje etapy kompilowania i łączenia. Skonfiguruj środowisko tak, jak w kroku “3” na stronie 565 przed uruchomieniem komendy make.

```
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 używania komendy make zawiera publikacja z/OS UNIX System Services Programming Tools .

Działający

1. Zaloguj się do powłoki usług systemowych UNIX i przejdź do katalogu, w którym została zbudowana próba.
2. Skonfiguruj zmienną środowiskową STEPLIB w celu uwzględnienia zestawów danych produktu 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 produktu .NET

Program IBM MQ classes for .NET umożliwia programowi napisanego w środowisku programowania .NET nawiązanie połączenia z serwerem IBM MQ jako IBM MQ MQI client lub nawiązanie bezpośredniego połączenia z serwerem IBM MQ .

Jeśli używane są aplikacje, które korzystają z programu Microsoft .NET Framework i chcą skorzystać z udogodnień 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 572.

V 9.1.1 From IBM MQ 9.1.1, IBM MQ supports .NET Core for applications in Windows environments. Aby uzyskać więcej informacji, zapoznaj się z sekcją: “Instalowanie produktu IBM MQ classes for .NET Standard” na stronie 568.

V 9.1.2 From IBM MQ 9.1.2, IBM MQ supports .NET Core for applications in Linux environments.

V 9.1.4 From IBM MQ 9.1.4, IBM MQ .NET managed applications are able to automatically balance connections across clustered queue managers. Obsługiwane są zarówno biblioteki .NET Framework , jak i .NET Standard . Więcej informacji na ten temat zawiera sekcja Klastry jednolite i Automatyczne równoważenie aplikacji.

Zorientowany obiektowo interfejs IBM MQ .NET różni się od interfejsu MQI, ponieważ używa metod obiektów, a nie za pomocą czasowników MQI.

Proceduralny interfejs programistyczny aplikacji IBM MQ jest zbudowany wokół czasowników, takich jak na poniższej liście: .

```
      MQCONN, MQDISC, MQOPEN, MQCLOSE,
      MQINQ, MQSET, MQGET, MQPUT, MQSUB
```

Te czasowniki przyjmują, jako parametr, uchwyt do obiektu IBM MQ , na którym mają działać. Ponieważ produkt .NET jest zorientowany obiektowo, interfejs programistyczny .NET włącza tę runę. Program składa się z zestawu obiektów IBM MQ , które są używane przez wywołania metod na tych obiektach. Programy można pisać w dowolnym języku obsługiwanym przez produkt .NET.

Jeśli używany jest interfejs proceduralny, należy odłączyć się od menedżera kolejek za pomocą wywołania MQDISC (*Hconn*, CompCode, Reason), gdzie *Hconn* jest uchwytem menedżera kolejek.

W interfejsie .NET menedżer kolejek jest reprezentowany przez obiekt klasy MQQueueManager. Odłącz się od menedżera kolejek, wywołując metodę 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 produktu .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 można znaleźć w sekcji Klasy i interfejsy IBM MQ .NET.

Zanim możliwe będzie skompilowanie wszystkich napisanych aplikacji, konieczne jest zainstalowanie środowiska .NET . Instrukcje dotyczące instalowania produktu IBM MQ classes for .NET i środowiska .NET zawiera sekcja “Instalowanie produktu IBM MQ classes for .NET Framework” na stronie 572.

Pojęcia pokrewne

“Opcje łączenia produktu IBM MQ classes for .NET z menedżerem kolejek” na stronie 567
Istnieją trzy tryby nawiązywania połączenia IBM MQ classes for .NET z menedżerem kolejek. Należy wziąć pod uwagę, który typ połączenia najlepiej odpowiada wymaganiom użytkownika.

“Pisanie i wdrażanie programów IBM MQ .NET” na stronie 587
Aby użyć programu IBM MQ classes for .NET w celu uzyskania dostępu do kolejek produktu IBM MQ , należy pisać programy w dowolnym języku obsługiwanym przez produkt .NET , który zawiera wywołania, które umieszczają komunikaty w kolejkach produktu IBM MQ i pobierają komunikaty z nich.

“Tworzenie aplikacji dla składnika IBM MQ” na stronie 5
Istnieje możliwość tworzenia aplikacji w celu wysyłania i odbierania komunikatów oraz do zarządzania menedżerami kolejek i powiązanymi zasobami. Produkt IBM MQ obsługuje aplikacje napisane w wielu różnych językach i w różnych ramach.

Zadania pokrewne

Rozwiązywanie problemów z produktem IBM MQ.NET

“Tworzenie aplikacji Microsoft Windows Communication Foundation przy użyciu produktu IBM MQ” na stronie 1379
Niestandardowy kanał programu Microsoft Windows Communication Foundation (WCF) dla IBM MQ wysyła i odbiera komunikaty między klientami i usługami WCF.

Odsyłacze pokrewne

Przegląd techniczny

Pierwsze kroki z produktem IBM MQ classes for .NET

Program IBM MQ classes for .NET umożliwia programowi napisanego w środowisku programowania .NET nawiązanie połączenia z serwerem IBM MQ jako IBM MQ MQI client lub nawiązanie bezpośredniego połączenia z serwerem IBM MQ .

Opcje łączenia produktu IBM MQ classes for .NET z menedżerem kolejek

Istnieją trzy tryby nawiązywania połączenia IBM MQ classes for .NET z menedżerem kolejek. Należy wziąć pod uwagę, który typ połączenia najlepiej odpowiada wymaganiom użytkownika.

Połączenie powiązań klienta

Aby użyć produktu IBM MQ classes for .NET jako IBM MQ MQI client, można go zainstalować za pomocą IBM MQ MQI clientna komputerze z serwerem IBM MQ lub na osobnym komputerze. Połączenie z powiązaniami klienta może używać transakcji XA lub innych niż XA.

Połączenie powiązań serwera

W przypadku użycia w trybie powiązań serwera produkt IBM MQ classes for .NET korzysta z interfejsu API menedżera kolejek, a nie komunikując się za pośrednictwem sieci. Zapewnia to lepszą wydajność aplikacji IBM MQ niż korzystanie z połączeń sieciowych.

Aby można było korzystać z połączenia powiązań, należy zainstalować produkt IBM MQ classes for .NET na serwerze IBM MQ .

Połączenie z klientem zarządzanym

Połączenie wykonane w tym trybie łączy się jako klient IBM MQ z serwerem IBM MQ działającym na komputerze lokalnym lub zdalnym.

IBM MQ classes for .NET łączące się w tym trybie pozostają w kodzie zarządzanym .NET i nie wywołują żadnych połączeń z usługami rodzimymi. Więcej informacji na temat kodu zarządzanego można znaleźć w dokumentacji produktu Microsoft .

Istnieje wiele ograniczeń dotyczących korzystania z zarządzanego klienta. Więcej informacji na ten temat zawiera sekcja “Połączenia zarządzane przez klienta” na stronie 587.

Windows Linux V 9.1.1 Instalowanie produktu IBM MQ classes for .NET Standard

Produkty IBM MQ 9.1.1, IBM MQ classes for .NET Standard, w tym przykłady, są instalowane razem z produktem IBM MQ w systemie Windows. V 9.1.2 Z poziomu produktu IBM MQ 9.1.2produkt IBM MQ classes for .NET Standard jest również dostępny na platformach Linux . Istnieje wymaganie wstępne Microsoft.NET Core dla IBM MQ classes for .NET Standard.

Wymagania wstępne i instalacja

Najnowsza wersja produktu IBM MQ classes for .NET Standard jest instalowana domyślnie jako część standardowej instalacji produktu IBM MQ w opcji *Java and .NET Messaging and Web Services* (Przesyłanie komunikatów języka Java i .NET).

W systemie IBM MQ 9.1.1produkt IBM MQ classes for .NET Standard jest dostępny tylko w systemie Windows .

V 9.1.2 W produkcie IBM MQ 9.1.2produkt IBM MQ classes for .NET Standard jest dostępny na platformach Linux oraz w produkcie Windows.

Aby uruchomić produkt IBM MQ classes for .NET Standard, należy zainstalować produkt Microsoft .NET Core.

Windows Więcej informacji na temat wymagań wstępnych i instalacji w systemie Windows:

- Informacje na temat wymaganego wstępnie oprogramowania do uruchamiania produktu IBM MQ classes for .NET Standardzawiera sekcja Wymagania dla produktu IBM MQ classes for .NET.
- Instrukcje instalacji znajdują się w sekcji Instalowanie serwera IBM MQ w systemie Windows lub Instalowanie klienta IBM MQ w systemach Windows .

Linux V 9.1.2 Więcej informacji na temat wymagań wstępnych i instalacji w systemie Linux:

- Informacje na temat wymaganego wstępnie oprogramowania do uruchamiania produktu IBM MQ classes for .NET Standardzawiera sekcja Wymagania dla produktu IBM MQ classes for .NET.

- Instrukcje dotyczące instalacji rpm-a zawiera sekcja Instalowanie klienta IBM MQ w systemach Linux.
- W przypadku systemu Linux Ubuntu, korzystając z pakietów Debian , należy zapoznać się z Instalowanie klienta IBM MQ w systemach Linux.

V 9.1.4 Z poziomu produktu IBM MQ 9.1.4biblioteka IBM MQ classes for .NET Standard `amqmdnetstd.dll`jest dostępna do pobrania z repozytorium produktu NuGet . Aby uzyskać więcej informacji, zapoznaj się z sekcją: “Pobieranie produktu IBM MQ classes for .NET Standard z repozytorium platformy NuGet” na stronie 571.

`amqmdnetstd.dll` biblioteka

Windows W produkcie IBM MQ 9.1.1biblioteka `amqmdnetstd.dll` jest dostępna dla obsługi produktu .NET Standard w systemie Windows. Dostarczane są również przykładowe aplikacje, w tym pliki źródłowe; patrz “Przykładowe aplikacje dla produktu .NET” na stronie 573.

Linux V 9.1.2 W produkcie IBM MQ 9.1.2biblioteka `amqmdnetstd.dll` jest również dostępna w systemie Linux. Biblioteka jest instalowana w produkcie `/&MQINSTALL_PATH&/lib64` `path` , gdy klient IBM MQ jest zainstalowany w systemie Linux. Przykłady produktu .NET znajdują się w katalogu `&MQINSTALL_PATH&/samp/dotnet/samples/cs/core/base`.

Każda biblioteka zbudowana w oparciu o specyfikację Microsoft .NET Standard może być używana do tworzenia aplikacji produktu .NET Framework , a także do tworzenia aplikacji produktu .NET Core .

Ostrzeżenie: Biblioteka `amqmdnet.dll` dla .NET Framework jest nadal dostarczana, ale ta biblioteka jest stabilna; oznacza to, że żadne nowe funkcje nie zostaną do niego wprowadzone.

W przypadku dowolnej z najnowszych składników należy przeprowadzić migrację do biblioteki produktu `amqmdnetstd.dll` . Można jednak nadal używać biblioteki `amqmdnet.dll` w wydaniach IBM MQ 9.1 Long Term Support lub Continuous Delivery .

Komenda `dspmqver`

W programie IBM MQ 9.1.1można użyć komendy **`dspmqver`** , aby wyświetlić informacje o wersji i kompilacji dla komponentu .NET Core .

Opcje IBM MQ classes for .NET Framework i IBM MQ classes for .NET Standard

W poniższej tabeli znajduje się lista funkcji produktu IBM MQ 9.1.1, z których można korzystać w produkcie IBM MQ classes for .NET Framework i IBM MQ classes for .NET Standard.

Tabela 74. Różnice w obsłudze funkcji między produktami IBM MQ classes for .NET Framework i IBM MQ classes for .NET Standard

Funkcja	IBM MQ classes for .NET Framework	IBM MQ classes for .NET Standard
Nazwy klas (interfejsy API)	Wszystkie klasy pozostają takie same w każdej sieci.	Wszystkie klasy pozostają takie same w każdej sieci.
System operacyjny	Windows	Windows Kontenery Docker V 9.1.2 Linux V 9.1.3 MacOS

Tabela 74. Różnice w obsłudze funkcji między produktami IBM MQ classes for .NET Framework i IBM MQ classes for .NET Standard (kontynuacja)

Funkcja	IBM MQ classes for .NET Framework	IBM MQ classes for .NET Standard
Plik `app.config` (plik konfiguracyjny do włączania śledzenia w kliencie redystrybuowanym)	Plik `app.config` jest używany do włączania śledzenia dla pakietu redystrybucyjnego.	`app.config` nie jest obsługiwany w .NET Standard. Konieczne jest użycie zmiennych środowiskowych w miejsce `app.config`.
Śledzenie	Komenda **`strmqtrc`** i plik `app.config` (w przypadku klientów podlegających redystrybucji) służą do włączania śledzenia.	Komenda **`strmqtrc`**. Do włączania śledzenia dla pakietu podlegającego redystrybucji służą zmienne środowiskowe. W przypadku systemu IBM MQ .NET zmienna środowiskowa **`MQDOTNET_TRACE_ON`** służy 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.
Tryby transportu	Zarządzany, Niezarządzany i Powiązania.	Zarządzany
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. V 9.1.2 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.

Tabela 74. Różnice w obsłudze funkcji między produktami IBM MQ classes for .NET Framework i IBM MQ classes for .NET Standard (kontynuacja)		
Funkcja	**IBM MQ classes for .NET Framework**	**IBM MQ classes for .NET Standard**
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 Windows (identyfikatory):

Uwierzytelnianie na poziomie domeny nie jest obsługiwane dla klas IBM MQ .NET Standard . Identyfikator zalogowanego użytkownika jest używany do uwierzytelniania.

Oprócz zmiennej środowiskowej **MQDOTNET_TRACE_ON** używanej do włączania śledzenia w systemie IBM MQ .NET Standardmożna używać także IBM MQ classes for .NET Standard zmiennych środowiskowych, w tym **MQERRORPATH**, **MQLOGLEVEL**, **MQSERVER**, i tak dalej, które są używane w przypadku produktu IBM MQ classes for .NET Framework. Te zmienne działają w taki sam sposób, jak w przypadku produktów IBM MQ classes for .NET Standard i IBM MQ classes for .NET Framework.

W programie IBM MQ classes for .NET Standardzmienna środowiskowa **MQDOTNET_TRACE_ON** sprawdza, czy katalog śledzenia produktu 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.

Więcej informacji na temat zmiennych, które są używane do śledzenia, w tym **MQTRACEPATH** i **MQTRACELEVEL**, zawiera sekcja “Korzystanie z autonomicznego klienta IBM MQ .NET” na stronie 619 .

Tworzenie aplikacji IBM MQ .NET Core w systemie MacOS

V 9.1.3

From IBM MQ 9.1.3, IBM MQ .NET Core applications can be developed on MacOS.

The IBM MQ .NET libraries are not packaged with the MacOS toolkit so you must copy them from a Windows or Linux IBM MQ client on to MacOS. Te biblioteki mogą być następnie używane do tworzenia aplikacji produktu IBM MQ .NET Core w systemie MacOS.

Po opracowaniu te aplikacje mogą być obsługiwane w środowiskach Windows lub Linux .

Pojęcia pokrewne

“Korzystanie z serwera IBM MQ classes for XMS .NET Standard” na stronie 629
Sposób użycia języka XMS z produktem Microsoft .NET Standardoraz różnice między używaniem języka IBM MQ classes for XMS .NET Framework i języka IBM MQ classes for XMS .NET Standard. Istnieje wymaganie wstępne produktu Microsoft.NET Core dla systemu IBM MQ classes for XMS .NET Standard.

Windows Linux V 9.1.4 ***Pobieranie produktu IBM MQ classes for .NET Standard z repozytorium platformy NuGet***

W serwisie IBM MQ 9.1.4produkt IBM MQ classes for .NET Standard 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 (DLL), inne pliki powiązane z tym kodem oraz opisowy manifest zawierający informacje takie jak numer wersji pakietu.

Z serwisu IBM MQ 9.1.4można pobrać pakiet produktu `IBMMQDotnetClient` NuGet , który zawiera bibliotekę `amqmdnetstd.dll` , z galerii NuGet , która jest centralnym repozytorium pakietów używanym przez wszystkich autorów i konsumentów pakietów.

Istnieją trzy sposoby pobierania pakietu `IBMMQDotnetClient` :

- Za pomocą komendy Microsoft Visual Studio. 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.

Zadania pokrewne

"Pobieranie produktu IBM MQ classes for XMS .NET Standard z repozytorium produktu NuGet" na stronie 631
From IBM MQ 9.1.4, the IBM MQ classes for XMS .NET Standard are available for downloading from the NuGet repository, so that they can be easily consumed by .NET Developers.

Odsyłacze pokrewne

Informacje licencyjne dotyczące programu IBM MQ Client for .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 programu Microsoft.NET Framework w systemie Windows.

Najnowsza wersja produktu IBM MQ classes for .NET Framework jest instalowana domyślnie jako część standardowej instalacji produktu IBM MQ w opcji *Java and .NET Messaging and Web Services* (Przesyłanie komunikatów języka Java i .NET). Instrukcje dotyczące instalowania znajdują się w sekcji Instalowanie serwera IBM MQ w systemie Windows lub Instalowanie klienta IBM MQ w systemach Windows.

V 9.1.0 From IBM MQ 9.1, to run IBM MQ classes for .NET Framework you must install Microsoft.NET Framework V4.5.1 or later.

Istniejące aplikacje skompilowane za pomocą programu Microsoft.NET Framework V3.5 można uruchamiać bez ponownego kompilowania, dodając następujący znacznik w pliku `app.config` aplikacji:

```
<configuration>
 <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5.1"/>
 </startup>
</configuration>
```

Uwaga: Jeśli produkt Microsoft .NET Framework V4.5.1 lub nowszy nie jest zainstalowany przed zainstalowaniem produktu IBM MQ, instalacja produktu IBM MQ będzie kontynuowana bez błędów, ale produkt IBM MQ classes for .NET nie jest dostępny. Jeśli produkt.NET Framework jest zainstalowany po zainstalowaniu produktu IBM MQ, zespoły IBM MQ.NET muszą być zarejestrowane, uruchamiając skrypt *WMQInstallDir*`\bin\amqiRegisterdotNet.cmd` , gdzie *WMQInstallDir* to katalog, w którym zainstalowano produkt IBM MQ . Ten skrypt zainstaluje wymagane zespoły w globalnej pamięci podręcznej zespołu (Global Assembly Cache-GAC). W katalogu %TEMP% tworzony jest zestaw plików produktu `amqi*.log` , które rejestrują działania, które zostały wykonane. Nie jest konieczne ponowne uruchomienie skryptu `amqiRegisterdotNet.cmd` , jeśli produkt .NET jest aktualizowany do wersji v4.5.1 lub nowszej z wcześniejszej wersji, na przykład z wersji .NET V3.5.

W środowisku z wieloma instalacją, jeśli produkt IBM MQ classes for .NET został wcześniej zainstalowany jako pakiet wsparcia, nie można zainstalować produktu IBM MQ , o ile nie zostanie zdeinstalowany pakiet obsługi. Opcja IBM MQ classes for .NET , która jest instalowana razem z produktem IBM MQ , zawiera te same funkcje, co pakiet wsparcia.

Dostarczane są również przykładowe aplikacje, w tym pliki źródłowe; patrz “Przykładowe aplikacje dla produktu .NET” na stronie 573.

Informacje na temat korzystania z niestandardowego kanału IBM MQ dla systemu Microsoft WCF z produktem .NETmożna znaleźć w sekcji “Tworzenie aplikacji Microsoft Windows Communication Foundation przy użyciu produktu IBM MQ” na stronie 1379 .

Pojęcia pokrewne

“Instalowanie produktu IBM MQ classes for .NET Standard” na stronie 568
Produkty IBM MQ 9.1.1, IBM MQ classes for .NET Standard, w tym przykłady, są instalowane razem z produktem IBM MQ w systemie Windows. V 9.1.2 Z poziomu produktu IBM MQ 9.1.2produkt IBM MQ classes for .NET Standard jest również dostępny na platformach Linux . Istnieje wymaganie wstępne Microsoft.NET Core dla IBM MQ classes for .NET Standard.

Przykładowe aplikacje dla produktu .NET

Aby uruchomić własne aplikacje produktu .NET , należy użyć instrukcji dla programów weryfikujących, podstawiając nazwę aplikacji w miejsce aplikacji przykładowych.

Dostarczane są następujące przykładowe aplikacje:

- Aplikacja umieszczanie komunikatów
- Aplikacja pobierania komunikatów
- Aplikacja "hello world"
- Aplikacja publikowania/subskrypcji
- Aplikacja używała właściwości komunikatu

Wszystkie te przykładowe aplikacje są dostarczane w języku C#, a niektóre z nich są również dostarczane w języku C++ i Visual Basic. Aplikacje można pisać w dowolnym języku obsługiwanym przez produkt .NET.

Program SPUT programu "Put message" (nmqsput.cs, mmqsput.cpp, vmqsput.vb)
Ten program pokazuje, jak umieścić komunikat w nazwanej kolejce. Program ma trzy parametry:

- Nazwa kolejki (wymagane), na przykład SYSTEM.DEFAULT.LOCAL.QUEUE
- Nazwa menedżera kolejek (opcjonalnie)
- Definicja kanału (opcjonalnie), na przykład SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

Jeśli nie zostanie podana nazwa menedżera kolejek, menedżer kolejek zostanie domyślnie ustawiony na domyślny lokalny menedżer kolejek. Jeśli kanał jest zdefiniowany, ma taki sam format, jak w przypadku zmiennej środowiskowej MQSERVER.

Program SGET programu "Get message" (nmqsget.cs, mmqsget.cpp, vmqsget.vb)
Ten program pokazuje, jak uzyskać komunikat z nazwanej kolejki. Program ma trzy parametry:

- Nazwa kolejki (wymagane), na przykład SYSTEM.DEFAULT.LOCAL.QUEUE
- Nazwa menedżera kolejek (opcjonalnie)
- Definicja kanału (opcjonalnie), na przykład SYSTEM.DEF.SVRCONN/TCP/hostname(1414)

Jeśli nie zostanie podana nazwa menedżera kolejek, menedżer kolejek zostanie domyślnie ustawiony na domyślny lokalny menedżer kolejek. Jeśli kanał jest zdefiniowany, ma taki sam format, jak w przypadku zmiennej środowiskowej MQSERVER.

Program "Hello World" (nmqwrld.cs, mmqwrld.cpp, vmqwrld.vb)
Ten program pokazuje, jak umieścić i otrzymać wiadomość. Program ma trzy parametry:

- Nazwa kolejki (opcjonalnie), 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 jest podana, wartością domyślną jest SYSTEM.DEFAULT.LOCAL.QUEUE. Jeśli nie zostanie podana nazwa menedżera kolejek, menedżer kolejek zostanie domyślnie ustawiony na domyślny lokalny menedżer kolejek.

Program "Publish/subscribe" (MQPubSubSample.cs)
Ten program pokazuje, jak używać publikowania/subskrypcji produktu IBM MQ . Jest on dostarczany tylko w C#. Program ma dwa parametry:

- Nazwa menedżera kolejek (opcjonalnie)
- Definicja kanału (opcjonalnie)

Program "Właściwości komunikatu" (MQMessagePropertiesSample.cs)
Ten program pokazuje, jak używać właściwości komunikatu. Jest on dostarczany tylko w C#. Program ma dwa parametry:

- Nazwa menedżera kolejek (opcjonalnie)
- Definicja kanału (opcjonalnie)

Instalację można zweryfikować, kompilując i uruchamiając te aplikacje.

Miejsca instalacji

Aplikacje przykładowe są instalowane w następujących lokalizacjach, zgodnie z językiem, w którym są zapisywane. *MQ_INSTALLATION_PATH* reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest 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

Zarządzany 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

Aby zbudować przykładowe aplikacje, dla każdego języka dostarczany jest plik wsadowy.

C#

MQ_INSTALLATION_PATH\Tools\dotnet\samples\cs\bldcssamp.bat

Plik bldcssamp.bat zawiera wiersz dla każdej próbki, która jest niezbędna do zbudowania tego przykładowego programu:

```
csc /t:exe /r:System.dll /r:amqmdnet.dll /lib: MQ_INSTALLATION_PATH\bin
/out:nmqwrld.exe nmqwrld.cs
```

Zarządzany C++

MQ_INSTALLATION_PATH\Tools\dotnet\samples\mcp\bldmcpsamp.bat

Plik bldmcpsamp.bat zawiera wiersz dla każdej próbki, która jest niezbędna do zbudowania tego przykładowego programu:

```
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 kompilowania:

```
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 każdej próbki, która jest niezbędna do zbudowania tego przykładowego programu:

```
vbc /r:System.dll /r: MQ_INSTALLATION_PATH\bin\amqmdnet.dll /out:vmqwrld.exe vmqwrld.vb
```

Samples for using IBM MQ with Microsoft .NET Core

V 9.1.1

From IBM MQ 9.1.1, IBM MQ supports .NET Core for IBM MQ .NET applications in Windows environments. IBM MQ classes for .NET Standard, w tym przykłady, są instalowane domyślnie jako część standardowej instalacji produktu IBM MQ .

Przykładowe aplikacje dla programu IBM MQ .NET są zainstalowane w produkcie `&MQINSTALL_PATH&/samp/dotnet/samples/cs/core/base`. Dostępny jest również skrypt, który może być używany do kompilowania przykładów.

Przykłady można zbudować, korzystając z dostarczonych plików produktu `build.bat` . Dla każdej próbki w następującym miejscu w systemie Windowsznajduje się jeden `build.bat` :

- `MQ\tools\dotnet\samples\cs\core\base\SimpleGet`
- `MQ\tools\dotnet\samples\cs\core\base\SimplePut`

Linux V 9.1.2 Z poziomu produktu IBM MQ 9.1.2produkt IBM MQ obsługuje również moduł .NET Core dla aplikacji w środowiskach Linux .

Więcej informacji na temat używania produktu IBM MQ z Microsoft .NET Core zawiera sekcja “Instalowanie produktu IBM MQ classes for .NET Standard” na stronie 568.

Konfigurowanie menedżera kolejek w celu akceptowania połączeń klientów TCP/IP

Skonfiguruj menedżer kolejek w taki sposób, aby akceptować przychodzące żądania połączeń od klientów.

O tym zadaniu

W tym zadaniu wyjaśniono podstawowe czynności związane z konfigurowaniem menedżera kolejek w celu akceptowania połączeń klientów TCP/IP. W przypadku systemu produkcyjnego konieczne jest również uwzględnienie wpływu na bezpieczeństwo 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 do użytku wyłącznie w środowisku przestrzeni prywatnej, ponieważ nie uwzględnia żadnych konsekwencji związanych z bezpieczeństwem. 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; parametr *qmname* nie jest wymagany dla domyślnego menedżera kolejek, a numer portu *numer_portu* nie jest wymagany, jeśli jest używany domyślny numer portu (1414).

Transakcje rozproszone w produkcie .NET

Transakcje rozproszone lub transakcje globalne umożliwiają aplikacjom klienckim kilka różnych źródeł danych na dwóch lub więcej systemach sieciowych w jednej transakcji.

W transakcjach rozproszonych menedżer transakcji koordynuje transakcję i zarządza nią wśród dwóch lub większej liczby menedżerów zasobów.

Transakcje mogą być procesem zatwierdzania jednofazowego lub dwufazowego. Zatwierdzanie jednofazowe to proces, w którym tylko jeden menedżer zasobów uczestniczy w procesie zatwierdzania transakcji, a proces zatwierdzania dwufazowego jest w przypadku, gdy uczestniczy w transakcji 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 ze wszystkich menedżerów zasobów wywołanie zatwierdzenia jest wykonywane. W przeciwnym razie dochodzi do wycofania zmian w całej transakcji. Więcej informacji na ten temat zawiera sekcja Zarządzanie transakcjami i obsługa . Kierownicy zasobów powinni informować kierowników transakcji o swoim udziale w transakcji. Gdy menedżer zasobów poinformuje menedżera transakcji o swoim udziale, menedżer zasobów pobiera oddzwonienia od menedżera transakcji, gdy transakcja ma zostać zatwierdzona lub wycofana.

Klasy IBM MQ .NET obsługują już transakcje rozproszone w połączeniach niezarządzanych i w trybie powiązań między serwerami. W tych trybach klasy IBM MQ .NET delegują wszystkie jego wywołania do rozszerzonego klienta transakcji w języku C, który zarządza przetwarzaniem transakcji w imieniu serwera .NET.

Klasy IBM MQ.NET obsługują teraz transakcje rozproszone w trybie zarządzanym, w którym klasy IBM MQ .NET korzystają z przestrzeni nazw System.Transactions dla obsługi transakcji rozproszonych. Infrastruktura System.Transactions sprawia, że programowanie transakcyjne jest proste i wydajne, dzięki obsłudze transakcji zainicjowanych we wszystkich menedżerach zasobów, w tym w produkcie IBM MQ. Aplikacja IBM MQ .NET może umieszczać i pobierania komunikatów przy użyciu niejawnego programowania transakcji .NET lub jawnego modelu programowania transakcji. W transakcjach niejawnych granice transakcji są tworzone przez program użytkowy, który podejmuje decyzję o zatwierdzeniu, wycofaniu (w przypadku transakcji jawnych) lub zakończeniu transakcji. W transakcjach jawnych konieczne jest jawne określenie, czy transakcja ma zostać zatwierdzona, wycofana, czy zakończona.

IBM MQ.NET uses Microsoft distributed transaction coordinator (MS DTC) as the transaction manager, which coordinates and manages the transaction between multiple resource managers. IBM MQ jest używany jako menedżer zasobów. Należy pamiętać, że nie można używać protokołu TLS z transakcjami XA. Należy użyć tabeli definicji kanału klienta. Więcej informacji na ten temat zawiera sekcja Korzystanie z rozszerzonego klienta transakcyjnego z kanałami TLS.

Program IBM MQ.NET jest zgodny z modelem X/Open Distributed Transaction Processing (DTP). Model X/Open Distributed Transaction Processing jest rozproszonym modelem przetwarzania transakcji zaproponowanym przez Open Group, konsorcjum dostawców. Model ten jest standardem wśród większości komercyjnych dostawców w zakresie przetwarzania transakcyjnego i domen baz danych. Większość produktów do zarządzania transakcjami komercyjnymi obsługuje model X/DTP.

Tryby transakcji

- “Transakcje rozproszone w trybie zarządzanym .NET” na stronie 578
- Rozproszone transakcje dla trybu niezarządzanego

Koordynowanie transakcji w różnych scenariuszach

- Połączenie może uczestniczyć w kilku transakcjach, ale tylko jedna transakcja jest aktywna w dowolnym momencie.
- Podczas transakcji MQQueueManager.Wywołanie odłączenia zostało uhonorowane. W takim przypadku transakcja jest proszona o wycofanie zmian.

- Podczas transakcji honorowane jest wywołanie MQQueue.Close lub MQTopic.Close . W tym przypadku transakcja jest proszona o wycofanie zmian.
- Granice transakcji są tworzone przez program użytkowy, który decyduje, kiedy zatwierdzić, wycofać (w przypadku transakcji jawnych) lub zakończyć (dla transakcji niejawnych) transakcję.
- Jeśli aplikacja kliencka przerwie działanie podczas transakcji z nieoczekiwanym błędem przed wywołaniem funkcji Put lub Get w wywołaniu kolejki lub tematu, transakcja jest wycofana i zgłaszany jest wyjątek MQException.
- Jeśli kod przyczyny MQCC_FAILED jest zwracany podczas wywoływania operacji umieszczania lub pobierania w kolejce lub w wywołaniu tematu, zgłaszany jest wyjątek MQException z kodem przyczyny, a transakcja jest wycofana. Jeśli menedżer transakcji wydał już żądanie przygotowania, program IBM MQ .NET zwraca żądanie przygotowania, wymuszając wycofywanie transakcji. Następnie menedżer transakcji DTC powoduje wycofanie zmian w bieżącej pracy ze wszystkimi menedżerami zasobów w bieżących transakcjach otoczenia.
- W przypadku transakcji obejmującej wiele menedżerów zasobów, jeśli jakiś powód środowiskowy powoduje, że wywołanie funkcji Put lub Get jest nieokreślone, menedżer transakcji oczekuje na określony czas. Po zakończeniu tego czasu wycofywanie wszystkich bieżących prac ze wszystkimi menedżerami zasobów w bieżących transakcjach otoczenia jest przyczyną wycofania wszystkich bieżących prac. Jeśli ten nieokreślony czas oczekiwania zostanie przekroczony podczas fazy przygotowania, menedżer transakcji może przekroczenie limitu czasu lub wywołać wątpliwe wywołanie zasobu, w którym to przypadku transakcja jest wycofana.
- Aplikacje korzystające z transakcji muszą umieścić lub pobrać komunikaty w katalogu SYNC_POINT. Jeśli komunikat Put lub Get jest generowany w kontekście transakcyjnym, który nie znajduje się w obszarze SYNC_POINT, wywołanie nie powiedzie się i zostanie zwrócony kod przyczyny MQRC_UNIT_OF_WORK_NOT_STARTED.

Różnice behawioralne między obsługą transakcji klienta zarządzanego i niezarządzanego przy użyciu przestrzeni nazw Microsoft.NET System.Transactions

Transakcje zagnieżdżone mają TransactionScope w innym elemencie TransactionScope

- W pełni zarządzany klient IBM MQ .NET obsługuje zagnieżdżony TransactionScope
- Niezarządzany klient IBM MQ .NET nie obsługuje zagnieżdżonego TransactionScope

Transakcje zależne od System.Transactions

- W pełni zarządzany klient IBM MQ .NET obsługuje funkcję transakcji zależnych udostępnianą przez System.Transactions.
- Klient niezarządzany IBM MQ .NET nie obsługuje mechanizmu transakcji zależnych udostępnianego przez System.Transactions.

Przykłady produktów

Nowe przykłady produktów `SimpleXAPuti SimpleXAGet` są dostępne pod `WebSphere MQ\tools\dotnet\samples\cs\base`. Przykłady to aplikacje typu C#, które demonstrują za pomocą operacji MQPUT i MQGET w ramach transakcji rozproszonych przy użyciu przestrzeni nazw SystemTransactions . Więcej informacji na temat tych przykładów można znaleźć w sekcji “Tworzenie prostych komunikatów put i get w ramach TransactionScope” na stronie 581 .

Transakcje rozproszone w trybie zarządzanym .NET

Klasy IBM MQ .NET używają przestrzeni nazw System.Transactions dla obsługi transakcji rozproszonych w trybie zarządzanym. W trybie zarządzanym MS DTC koordynuje i zarządza rozproszonymi transakcjami na wszystkich serwerach wymienionych w transakcji.

Klasy IBM MQ .NET udostępniają jawny model programistyczny oparty na klasie System.Transactions.Transaction i niejawny model programistyczny przy użyciu 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 transakcji .NET .

```
Using (TransactionScope scope = new TransactionScope ())
{
    Q.Put (putMsg,pmo);
    scope.Complete ();
}

Q.close();
qMgr.Disconect();}
```

Wyjaśnienie przepływu kodu dla transakcji niejawnej

Kod tworzy obiekt *TransactionScope* i umieszcza komunikat w zasięgu. Następnie wywołuje *Zakończone* , aby poinformować koordynatora transakcji o zakończeniu transakcji. Koordynator transakcji wydaje teraz *przygotowanie* i *zatwierdzanie* w celu zakończenia transakcji. Jeśli problem zostanie wykryty, wywoływana jest *wycofywanie zmian* .

Transakcja jawna

Poniższy kod opisuje sposób, w jaki aplikacja IBM MQ .NET umieszcza komunikaty przy użyciu jawnego modelu programowania transakcji w produkcie .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 dla transakcji jawnej

Fragment kodu tworzy transakcję przy użyciu klasy *CommitableTransaction* . Umieszcza komunikat w tym zasięgu, a następnie jawnie wywołuje komendę *commit* , aby zakończyć transakcję. Jeśli wystąpią jakieś problemy, *wycofywanie zmian* jest wywoływane.

Transakcje rozproszone w trybie niezarządzanym .NET

Klasy IBM MQ.NET obsługują połączenia niezarządzane (klient) przy użyciu rozszerzonego klienta transakcji oraz COM + /MTS jako koordynator transakcji, przy użyciu niejawnego lub jawnego modelu programowania transakcji. W trybie niezarządzanym klasy IBM MQ .NET delegują wszystkie jego wywołania do rozszerzonego klienta transakcji w języku C, który zarządza przetwarzaniem transakcji w imieniu serwera .NET.

Przetwarzanie transakcji jest kontrolowane przez zewnętrznego menedżera transakcji, koordynując globalną jednostkę pracy pod kontrolą interfejsu API menedżera transakcji. Czasowniki MQBEGIN, MQCMIT i MQBACK są niedostępne. IBM MQ Klasy produktu .NET prezentują tę obsługę za pośrednictwem niezarządzanego trybu transportu (klient C). Patrz sekcja Konfigurowanie menedżerów transakcji zgodnych z interfejsem XA .

System MTS jest oparty na systemie przetwarzania transakcyjnego (TP) w celu udostępnienia tych samych funkcji w produkcie Windows NT w wersji CICS, Tuxedo i na innych platformach. Po zainstalowaniu MTS oddzielna usługa jest dodawana do Windows NT o nazwie Microsoft Distributed Transaction Coordinator (MSDTC). MSDTC koordynuje transakcje, które obejmują oddzielne składnice danych lub zasoby. Do pracy wymagane jest, aby każda składnica danych zaimplementował własny, własny menedżer zasobów.

Produkt IBM MQ staje się zgodny z produktem MSDTC , implementując interfejs (interfejs menedżera zasobów zastrzeżonych), w którym zarządza odwzorowaniem wywołań interfejsu DTC XA na wywołania programu IBM MQ(X/Open). Produkt IBM MQ pełni rolę menedżera zasobów.

Gdy komponent, taki jak COM + żąda dostępu do IBM MQ, COM zwykle sprawdza odpowiedni obiekt kontekstu MTS, jeśli wymagana jest transakcja. Jeśli transakcja jest wymagana, COM informuje DTC i automatycznie uruchamia integracją transakcji IBM MQ dla tej operacji. Następnie COM współpracuje z danymi za pomocą 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. Gdy aplikacja wydaje komendę SetComplete, wywołanie sygnalizuje DTC, że aplikacja zakończyła transakcję, a DTC może wyprzeć proces zatwierdzania dwufazowego. Następnie DTC wywołuje wywołania MQMTS, które z kolei wywołują wywołania programu IBM MQ w celu zatwierdzenia lub wycofania transakcji.

Zapisywanie aplikacji IBM MQ .NET przy użyciu niezarządzanego klienta

Aby można było uruchomić w kontekście COM +, klasa .NET musi dziedziczyć z systemu System.EnterpriseServices.ServicedComponent. Reguły i zalecenia dotyczące tworzenia zespołów, które korzystają z obsługiwanych komponentów, są następujące:

Uwaga: Poniższe kroki są istotne tylko wtedy, gdy używany jest tryb System.EnterpriseServices .

- Klasa i metoda uruchamiana w COM + muszą być publiczne (nie ma klas wewnętrznych i nie są chronione ani statyczne).
- Atrybuty klasy i metody: atrybut TransactionOption określa poziom transakcji klasy, to znaczy, czy transakcje są wyłączone, obsługiwane lub wymagane. Atrybut AutoComplete w metodzie ExecuteUOW() instruuje COM +, aby zatwierdzić transakcję, jeśli nie został zgłoszony żaden nieobsługiwany wyjątek.
- Silne-nazywanie zespołu: Zespół musi być mocny-nazwany i zarejestrowany w globalnej pamięci podręcznej zespołu (GAC). Montaż jest rejestrowany w COM + wyraźnie lub przez opóźnione rejestrację po zarejestrowaniu się w GAC.
- Rejestrowanie zespołu w COM +: Przygotowanie zespołu do wystawiania na potrzeby klientów 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 COM +, korzystając z narzędzia instalatora usług .NET , regsvcs.exe. Zapoznaj się z biblioteką typów utworzoną przez program regasm.exe:

  ```
  Regsvcs /appname:UnmanagedToManagedXa /tlb:UnmanagedToManagedXa.tlb  UnmanagedToManagedXa.dll
  ```

- Montaż jest wdrażany w GAC, a później jest zarejestrowany w COM + przez leniwą rejestrację. Środowisko .NET zajmuje się rejestracją po uruchomieniu kodu po raz pierwszy.

Przykładowy przepływ kodu przy użyciu modelu System.EnterpriseServices i pliku System.Transactions z COM + są opisane 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 metody 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 komunikatów put i get w ramach TransactionScope

Przykładowe aplikacje C# produktu są dostępne w produkcie IBM MQ. Te proste aplikacje demonstrują wprowadzanie i pobieranie komunikatów w elemencie TransactionScope. Po zakończeniu zadania użytkownik będzie mógł umieścić i pobrać komunikaty z kolejki lub tematu.

Zanim rozpoczniesz

Usługa MSDTC musi być uruchomiona i włączona dla transakcji XA.

O tym zadaniu

Przykładem jest prosta aplikacja, `SimpleXAPut` i `SimpleXAGet`. Programy `SimpleXAPut` i `SimpleXAGet` to aplikacje C# dostępne w produkcie IBM MQ. Produkt `SimpleXAPut` demonstruje za pomocą wywołania MQPUT w ramach transakcji rozproszonych przy użyciu przestrzeni nazw SystemTransactions . `SimpleXAGet` demonstruje za pomocą wywołania MQGET w ramach transakcji rozproszonych przy użyciu przestrzeni nazw SystemTransactions .

`SimpleXAPut` znajduje się w `MQ\tools\dotnet\samples\cs\base`

Procedura

Aplikacje mogą być uruchamiane z parametrami wiersza komend z programu `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 jako `queue://queueName` , a dla tematu określić jako `topic://topicName`.

-host

Może to być nazwa hosta, taka jak localhost lub adres IP.

-port

Port, na którym działa menedżer kolejek.

-channel

Używany kanał połączenia. Wartością domyślną jest `SYSTEM.DEF.SVRCONN`

-transaction

Wynik transakcji, na przykład zatwierdzenie lub wycofanie zmian.

-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 w produkcie IBM MQ .NET XA przy użyciu trybu zarządzanego.

Przegląd

W przypadku przetwarzania transakcji rozproszonych transakcje mogą być pomyślnie zakończone. Ale mogą wystąpić sytuacje, w których transakcja może nie powieść się 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. Zapobieganie awariom transakcji nie jest możliwe. Rozproszony system transakcyjny musi być w stanie obsługiwać te niepowodzenia. Musi być w stanie wykryć i poprawić błędy w momencie ich wystąpienia. Ten proces jest znany jako Odtwarzanie transakcji.

Istotnym aspektem przetwarzania rozproszonego transakcji jest odzyskanie niekompletnych lub wątpliwych transakcji. Konieczne jest uruchomienie odtwarzania, ponieważ część konkretnej transakcji jest wstrzymana do czasu jej odtworzenia. Program Microsoft.NET z biblioteki klas System.Transactions udostępnia opcję odtwarzania niekompletnych/wątpliwych transakcji. Ta obsługa odtwarzania oczekuje, że Resource Manager będzie obsługiwał dzienniki transakcji i w razie potrzeby będzie uruchamiać odtwarzanie.

Model odzyskiwania

W modelu odtwarzania transakcji Microsoft .NET menedżer transakcji (System.Transactionslub Microsoft Distributed Transaction Coordinator (MS DTC) lub oba te elementy) inicjuje, koordynuje i steruje odtwarzaniem transakcji. Protokół OLE Tx (protokół Microsoft XA) oparty na Menedżerach zasobów udostępnia opcje służące do konfigurowania DTC do kierowania, koordynowania i kontrolowania dla nich odtwarzania. Aby to zrobić, menedżerowie zasobów muszą zarejestrować XA_Switch z MS DTC za pomocą rodzimego interfejsu.

Funkcja XA_Switch udostępnia punkty wejścia funkcji XA, takie jak xa_start, xa_end i xa_recover, w Resource Manager , do koordynatora transakcji rozproszonych.

Odtwarzanie przy użyciu koordynatora rozproszonego transakcji produktu Microsoft (DTC):

Microsoft Rozproszony koordynator transakcji udostępnia dwa rodzaje procesów odtwarzania.

Zimne odtwarzanie

Odtwarzanie na zimno jest wykonywane, jeśli proces menedżera transakcji zakończy się niepowodzeniem, 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 pracy jest wykonywane wtedy, gdy menedżer transakcji pozostaje włączony, gdy połączenie między menedżerem transakcji a menedżerem zasobów XA nie powiedzie się, ponieważ menedżer zasobów XA lub sieć nie powiodły się. Po awarii menedżer transakcji okresowo podejmuje próbę ponownego nawiązania połączenia z menedżerem zasobów XA. Po ponownym nawiązaniu połączenia menedżer transakcji inicjuje odtwarzanie XA.

Przestrzeń nazw System.Transactions udostępnia zarządzane implementację transakcji rozproszonych, które są oparte na programie MS DTC jako menedżer transakcji. Udostępnia on podobne funkcje, jak interfejs rodzimy w programie MS DTC, ale w pełni zarządzane środowisko. Jedyna różnica polega na odzyskiwaniu transakcji. System.Transactions oczekuje, że kierownicy zasobów będą prowadzić odtwarzanie samodzielnie, a następnie koordynować je z menedżerami transakcji (MS DTC). Resource Manager musi poprosić o odzyskanie konkretnej niekompletnej transakcji, a następnie menedżer transakcji akceptuje je i koordynuje w oparciu o rzeczywisty wynik tej konkretnej transakcji.

Proces odtwarzania transakcji dla IBM MQ .NET
W tej sekcji opisano sposób, w jaki transakcje rozproszone mogą być odtwarzane z klasami 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 są zgodne z podobną ścieżką. Informacje o odtwarzaniu transakcji są rejestrowane w kolejce systemowej o nazwie SYSTEM.DOTNET.XARECOVERY.QUEUE.

Odtwarzanie transakcji w programie IBM MQ .NET jest procesem dwuetapowym.

1. Rejestrowanie informacji o odtwarzaniu transakcji.
   - Dla każdej transakcji podczas fazy przygotowywania komunikat trwały zawierający informacje o odtwarzaniu jest dodawany do systemu SYSTEM.DOTNET.XARECOVERY.QUEUE.
   - Komunikat zostanie usunięty, jeśli wywołanie zatwierdzenia powiedzie się.
2. Odtwarzanie transakcji przy użyciu aplikacji monitorującego WmqDotnetXAMonitor.
   - WmqDotnetXAMonitor to zarządzana aplikacja .NET , która przetwarza komunikaty w systemie SYSTEM.DOTNET.XARECOVERY.QUEUE i odzyskuje niekompletne transakcje

   Jeśli agent MCA nie może umieścić komunikatu w kolejce docelowej, generuje raport o wyjątkach zawierający oryginalny komunikat i umieszcza go w kolejce transmisji, która ma zostać wysłana do kolejki odpowiedzi określonej w pierwotnym komunikacie. (Jeśli kolejka zwrotna znajduje się w tym samym menedżerze kolejek co agent MCA, komunikat jest umieszczany bezpośrednio w tej kolejce, a nie do kolejki transmisji).

SYSTEM.DOTNET.XARECOVERY.QUEUE

Jest to kolejka systemowa, w której przechowywane są informacje o odtwarzaniu transakcji niekompletnych transakcji. Ta kolejka jest tworzona podczas tworzenia menedżera kolejek.

Uwaga: Nie należy usuwać systemu SYSTEM.DOTNET.XARECOVERY.QUEUE .

Aplikacja WMQDotnetXAMonitor

IBM MQ .NET Aplikacja Monitor XA monitoruje dany menedżer kolejek i odzyskuje niepełne transakcje, jeśli takie istnieją. Następujące transakcje są uważane za transakcje niekompletne i są odzyskane:

Niekompletne transakcje

- Jeśli transakcja jest przygotowana, ale COMMIT nie została zakończona w okresie limitu czasu.
- Jeśli transakcja jest przygotowana, ale menedżer kolejek produktu IBM MQ został wyłączony.
- Jeśli transakcja jest przygotowana, ale menedżer transakcji został wyłączony.

  Aplikacja monitorującego musi być uruchomiona z tego samego systemu, w którym działa aplikacja kliencka IBM MQ .NET . Jeśli istnieją aplikacje działające na wielu systemach łączących się z tym samym menedżerem kolejek, aplikacja monitorującego musi być uruchomiona ze wszystkich systemów. Mimo że każdy komputer kliencki ma aplikację monitorującą działającą w celu odzyskania aplikacji, każdy monitor powinien być w stanie zidentyfikować komunikat, który odpowiada transakcji, którą koordynował lokalny MS DTC w bieżącym monitorze, tak aby mógł on ponownie zarejestrować i zakończyć działanie tego monitora.

Przypadki użycia odtwarzania transakcji dla IBM MQ .NET
Istnieje kilka różnych przypadków użycia, z których transakcje mogą wymagać odtworzenia.

- **IBM MQ Aplikacja używała pojedynczej instancji DTC i pojedynczej instancji menedżera kolejek:** w tym przypadku podczas nawiązywania połączenia z menedżerem kolejek i uruchamiania jednostki pracy (UoW) w ramach transakcji, a jeśli transakcja nie powiedzie się i stanie się niekompletna, aplikacja monitorujący odzyskuje transakcję i zakończy ją.

  W tym przypadku zostanie uruchomiona pojedyncza instancja aplikacji monitorującego, ponieważ pojedynczy menedżer kolejek jest powiązany z transakcjami.

- **Wiele aplikacji produktu IBM MQ korzystających z pojedynczej instancji DTC i jednej instancji menedżera kolejek:** w tym przypadku użycia jest więcej niż jedno aplikacje WMQ w pojedynczym DTC i wszystkie połączenia nawiązują połączenie z tym samym menedżerem kolejek i z uruchomionym UoW w ramach transakcji.

  Jeśli transakcje nie powiodą się i staną się niekompletne, aplikacja monitorujący odzyskuje je i uzupełnia transakcje dotyczące wszystkich aplikacji.

  W tym przypadku używana jest jedna aplikacja monitora, ponieważ jeden menedżer kolejek jest używany w transakcjach.
- **Wiele aplikacji produktu IBM MQ , wiele obiektów DTC, różne instancje menedżera kolejek:** w tym przypadku użycia jest więcej niż jedna aplikacja WMQ w różnych DTC (czyli każda aplikacja jest uruchomiona na innym komputerze) i łączy się z różnymi menedżerami kolejek.

  Jeśli wystąpi awaria, a transakcja stanie się niekompletna, aplikacja monitorującego sprawdza, czy TransactionManagerw tym komunikacie ma być używany do określenia adresu DTC. Jeśli wartość TransactionManagerpowoduje, że wartość jest zgodna z adresem DTC, pod którym monitor jest uruchomiony, następuje zakończenie odtwarzania, a w przeciwnym razie trwa wyszukiwanie aż do znalezienia komunikatu odpowiadającego jego DTC.

  W tym przypadku w przypadku każdego klienta (użytkownika lub komputera) będzie działać tylko jedna instancja aplikacji monitorującego, ponieważ każdy klient ma własny menedżer kolejek używany w transakcjach.
- **Wiele aplikacji produktu IBM MQ , wiele rekordów DTC i wiele takich samych instancji menedżerów kolejek:** w tym przypadku użycia jest więcej niż jedna aplikacja WMQ w różnych DTC (każda aplikacja działa na innym komputerze), a wszystkie połączenia nawiązują połączenie z tym samym menedżerem kolejek.

  Jeśli wystąpi awaria, a transakcja stanie się niekompletna, aplikacja monitorującego weryfikuje obiekt TransactionManagerGdziekolwiek w komunikacie w celu sprawdzenia, czy adres DTC i wartość są zgodne z DTC, pod którym monitor jest uruchomiony. Jeśli obie wartości są zgodne, kontynuuje wyszukiwanie, dopóki nie znajdzie komunikatu odpowiadającego jego DTC.

  W tym przypadku w przypadku każdego klienta (użytkownika lub komputera) będzie działać tylko jedna instancja aplikacji monitorującego, ponieważ każdy klient ma własne powiązanie menedżera kolejek używane w transakcjach.
- **Wiele aplikacji produktu IBM MQ , pojedyncze DTC, różne instancje menedżera kolejek:** w tym przypadku użycia jest więcej niż jedna aplikacja WMQ w pojedynczym DTC (czyli na komputerze, na którym działa więcej niż jedna aplikacja WMQ) i łącząca się z różnymi menedżerami kolejek.

  Jeśli transakcja nie powiedzie się i stanie się niekompletna, aplikacja monitoruj odzyskuje transakcję.

  W tym przypadku użycia będzie tyle instancji aplikacji monitorujących, uruchomionych jako menedżery kolejek, z których nawiązano połączenie, ponieważ każda aplikacja ma własny menedżer kolejek używany w transakcjach, a każde z nich musi być odzyskane.

  Uwaga: Jeśli aplikacja monitora nie jest uruchomiona w tle, można ją uruchomić.

Korzystanie z aplikacji WMQDotnetXAMonitor
Aplikacja WMQDotnetXAMonitor musi być uruchamiana ręcznie. Można ją uruchomić w dowolnym momencie. Można go uruchomić, gdy wyświetlane są komunikaty w systemie SYSTEM.DOTNET.XARECOVERY.QUEUE lub można ją zachować w tle, zanim wykonasz jakiekolwiek prace transakcyjne z aplikacjami napisanych przy użyciu klas IBM MQ .NET .

Aby uruchomić aplikację monitora, użyj następującej komendy:

```
WmqDotnetXAMonitor.exe -m QueueManagerName -n ConnectionName -c ChannelName -i
```

Gdzie:

- **-m *QueueManagerNazwa***
  Nazwa menedżera kolejek.

Opcjonalne

-n *ConnectionName*
Nazwa połączenia w formacie hosta (port). Nazwa połączenia może zawierać więcej niż jedną nazwę połączenia. Nazwy wielu połączeń muszą być podane w postaci listy rozdzielanej przecinkami, na przykład `localhost (1414), localhost (1415), localhost (1416)`. Aplikacja Monitor uruchamia odtwarzanie dla każdej z nazw połączeń podanych w rozdzielanej przecinkami liście.

-c *ChannelName*
Nazwa kanału.

-h
Ukończenie oddziału heurystycznego.

Opcjonalne

Aplikacja monitorującego wykonuje następujące działania:

1. Sprawdza głębokość kolejki SYSTEM.DOTNET.XARECOVERY.QUEUE w odstępie 100 sekund.
2. Jeśli głębokość kolejki jest większa niż zero, monitor XA przegląda kolejkę dla komunikatów i sprawdza, czy komunikat spełnia niekompletne kryteria transakcji.
3. Jeśli którekolwiek z komunikatów spełnia niekompletne kryteria transakcji, monitor go pobiera i pobiera informacje o odtwarzaniu transakcji.
4. Następnie określa, czy informacje o odtwarzaniu odnoszą się do lokalnego DTC. Jeśli tak, to przechodzi w celu odzyskania transakcji. W przeciwnym razie zostanie ponownie wyświetlony następny komunikat.
5. Następnie wywołuje on wywołania do menedżera kolejek w celu odzyskania niekompletnej transakcji.

Ustawienia pliku konfiguracyjnego aplikacji WmqDotNETXAMonitor
Aby można było monitorować aplikację, dane wejściowe mogą być również udostępniane przy użyciu pliku konfiguracyjnego aplikacji. Przykładowy plik konfiguracyjny aplikacji jest dostarczany razem z programem IBM MQ .NET. Ten plik można modyfikować zgodnie z wymaganiami użytkownika.

Plik konfiguracyjny aplikacji ma najwyższy priorytet podczas uwzględniania wartości wejściowych. Jeśli wartości wejściowe są podane zarówno w wierszu komend, jak i w pliku konfiguracyjnym aplikacji, wówczas brane są pod uwagę wartości z konfiguracji aplikacji.

Przykładowy plik konfiguracyjny aplikacji.

```
<?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>
```

Dziennik aplikacji WmqDotNetXAMonitor
Aplikacja Monitor tworzy plik dziennika w katalogu aplikacji, który umożliwia rejestrowanie 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 uruchomieniu odtwarzania zostanie zarejestrowany identyfikator MessageId komunikatu o odtwarzaniu transakcji TransactionId niekompletnej transakcji i rzeczywisty wynik transakcji, jak na koordynację 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 użyć programu IBM MQ classes for .NET w celu uzyskania dostępu do kolejek produktu IBM MQ , należy pisać programy w dowolnym języku obsługiwanym przez produkt .NET , który zawiera wywołania, które umieszczają komunikaty w kolejkach produktu IBM MQ i pobierają komunikaty z nich.

Dokumentacja produktu IBM MQ zawiera informacje tylko w językach C#, C++ i Visual Basic.

Ta kolekcja tematów zawiera informacje pomocne przy pisaniu aplikacji w celu interakcji z systemami IBM MQ . Szczegółowe informacje na temat poszczególnych klas można znaleźć w sekcji Klasy i interfejsy IBM MQ .NET.

Różnice między połączeniami

Sposób, w jaki program IBM MQ.NET ma pewne zależności od trybów połączenia, które mają być używane.

Gdy produkt IBM MQ classes for .NET jest używany jako klient zarządzany, istnieje pewna liczba różnic ze standardowego pakietu IBM MQ MQI client, ponieważ niektóre opcje nie są dostępne dla klienta zarządzanego.

Program IBM MQ.NET określa, który typ połączenia ma być używany z ustawień określonych dla nazwy połączenia, nazwy kanału, wartości dostosowania NMQ_MQ_LIB i właściwości MQC.TRANSPORT_PROPERTY.

Połączenia zarządzane przez klienta

Gdy produkt IBM MQ classes for .NET jest używany jako klient zarządzany, istnieje pewna liczba różnic ze standardowego pakietu IBM MQ MQI client.

Następujące opcje nie są dostępne dla klienta zarządzanego:

- Kompresja kanału
- Łączenie wyjścia kanału

W przypadku próby użycia tych opcji z klientem zarządzanym zostanie zwrócony wyjątek MQException. Jeśli błąd zostanie wykryty na końcu połączenia klienta, zostanie użyty kod przyczyny MQRC_ENVIRONMENT_ERROR. Jeśli zostanie on 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 zarządzanego klienta. Sprawdź, czy w tabeli definicji kanału klienta (CCDT) nie określono żadnych niepoprawnych wyjść kanału.

Nazwa wyjścia kanału zarządzanego może mieć długość do 999 znaków. Jeśli jednak do określenia nazwy wyjścia kanału zostanie użyta wartość CCDT, to będzie ona ograniczona do 128 znaków.

Komunikacja jest obsługiwana tylko przez protokół TCP/IP.

Po zatrzymaniu menedżera kolejek za pomocą komendy **endmqm** kanał połączenia serwera z klientem zarządzanym .NET może zająć więcej czasu niż kanały połączenia serwera z innymi klientami.

Jeśli parametr *NMQ_MQ_LIB* jest ustawiony na wartość `managed` , aby można było używać procedur diagnostycznych systemu zarządzanego IBM MQ , nie jest obsługiwana żadna z parametrów -i, -p, -s, -b lub -c komendy **strmqtrc** .

Zarządzana aplikacja .NET używała transakcji XA nie będzie działać z menedżerem kolejek produktu z/OS . Klient .NET zarządzany przy próbie nawiązania połączenia z menedżerem kolejek produktu z/OS kończy się niepowodzeniem z błędem, MQRC_UOW_ENLISTMENT_ERROR (mqrc=2354) w wywołaniu MQOPEN. Jednak aplikacja .NET zarządzana za pomocą transakcji XA będzie działać z rozproszonym menedżerem kolejek.

Definiowanie typu połączenia, który ma być używany

Typ połączenia jest określany na podstawie ustawienia 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)
```

- W tym celu należy ustawić 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).
- W tym celu należy ustawić 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)
```

- W tym celu należy ustawić właściwość MQC.CHANNEL_PROPERTY w postaci tabeli mieszającej w konstruktorze MQQueueManager :

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- Jako jawna wartość MQEnvironment

```
MQEnvironment.Channel
```

- W tym celu należy ustawić właściwość MQC.CHANNEL_PROPERTY w tabeli mieszającej MQEnvironment.properties .

Właściwość transportu można określić w następujący sposób:

- W tym celu należy ustawić właściwość MQC.TRANSPORT_PROPERTY we wpisie tabeli mieszającej w konstruktorze MQQueueManager :

```
public MQQueueManager(String queueManagerName, Hashtable properties)
```

- W tym celu należy ustawić właściwość MQC.TRANSPORT_PROPERTY w tabeli hashtable 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 -połączenie jako klient XA
MQC.TRANSPORT_MQSERIES_MANAGED -połączenie jako klient zarządzany bez interfejsu XA

Wartość dostosowania NMQ_MQ_LIB można ustawić w taki sposób, aby jawnie wybrać typ połączenia, jak pokazano w poniższej tabeli.

Wartość 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
zarządzany	Połącz jako klient zarządzany inny niż XA

Uwaga: Wartości parametrów mqic32.dll i mqic32xa.dll są akceptowane jako synonimy plików mqic.dll i mqicxa.dll w celu zapewnienia zgodności z wcześniejszymi wersjami. Jednak pliki mqm.dll i mqm.pdb są tylko częścią pakietu klienta począwszy od wersji IBM WebSphere MQ 7.1 .

Jeśli zostanie wybrany typ połączenia, który nie jest dostępny w danym środowisku, na przykład określony został plik mqic32xa.dll i nie ma obsługi interfejsu XA, program IBM MQ.NET zgłosi wyjątek.

Ustawienie wartości NMQ_MQ_LIB na "managed" powoduje, że klient używa zarządzanych testów diagnostycznych problemów IBM MQ , konwersji danych .NET i innych zarządzanych niskopoziomowych funkcji IBM MQ .

Wszystkie pozostałe wartości dla właściwości NMQ_MQ_LIB powodują, że proces .NET korzysta z niezarządzanych testów diagnostycznych i konwersji danych IBM MQ , a także innych niezarządzanych niskopoziomowych funkcji IBM MQ (przy założeniu, że w systemie jest zainstalowany serwer IBM MQ MQI client lub serwer).

Program IBM MQ.NET wybiera typ połączenia w następujący sposób:

1. Jeśli MQC.TRANSPORT_PROPERTY , która łączy się z wartością MQC.TRANSPORT_PROPERTY.

   Należy jednak pamiętać, że ustawienie MQC.TRANSPORT_PROPERTY do MQC.TRANSPORT_MQSERIES_MANAGED nie gwarantuje, że proces klienta jest uruchamiany. Nawet w przypadku tego ustawienia klient nie jest zarządzany w następujących przypadkach:

   - Jeśli inny wątek w procesie nawiąże połączenie z MQC.TRANSPORT_PROPERTY jest ustawiona na wartość inną niż MQC.TRANSPORT_MQSERIES_MANAGED.
   - Jeśli wartość NMQ_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 serwer IBM MQ MQI client lub serwer).
2. Jeśli nazwa połączenia została określona bez nazwy kanału lub nazwa kanału została określona bez nazwy połączenia, to zgłasza błąd.

3. Jeśli określono zarówno nazwę połączenia, jak i nazwę kanału:
   - Jeśli wartość NMQ_MQ_LIB jest ustawiona na wartość mqic32xa.dll, łączy się ona jako klient XA.
   - Jeśli wartość NMQ_MQ_LIB jest ustawiona na zarządzany, łączy się ona jako klient zarządzany.
   - W przeciwnym razie łączy się ono jako klient inny niż XA.
4. Jeśli wartość NMQ_MQ_LIB jest określona, łączy się ona zgodnie z wartością NMQ_MQ_LIB.
5. Jeśli serwer IBM MQ jest zainstalowany, łączy się on jako serwer.
6. Jeśli zainstalowany jest produkt IBM MQ MQI client , łączy się on jako klient inny niż XA.
7. W przeciwnym razie łączy się on jako klient zarządzany.

Windows V 9.1.5 Korzystanie z szablonu projektu IBM MQ .NET

Z poziomu produktu IBM MQ 9.1.5klient IBM MQ .NET oferuje możliwość korzystania z szablonu projektu w celu ułatwienia tworzenia aplikacji produktu .NET Core .

Zanim rozpoczniesz

W systemie muszą być dostępne Microsoft Visual Studio 2017(lub nowsze) oraz .NET Core 2.1 (w systemie).

Należy skopiować szablon .NET z

```
&MQ_INSTALL_ROOT&\tools\dotnet\samples\cs\core\base\ProjectTemplates\IBMMQ.NETClientApp.zip
```

katalog do

```
&USER_HOME_DIRECTORY&\Documents\&Visual_Studio_Version&\Templates\ProjectTemplates
```

Katalog, w którym:

- *& MQ_INSTALL_ROOT* to katalog główny instalacji
- Katalog *& USER_HOME_DIRECTORY* jest katalogiem osobistym użytkownika.

Aby można było odebrać szablon, należy zatrzymać i zrestartować produkt Microsoft Visual Studio .

O tym zadaniu

Szablon projektu produktu .NET zawiera wspólny kod, którego można użyć w celu ułatwienia tworzenia aplikacji. Za pomocą wbudowanego kodu można nawiązać połączenie z menedżerem kolejek produktu IBM MQ , a następnie wykonać operację put lub get, modyfikując po prostu właściwości w kodzie wbudowanym.

Procedura

1. Otwórz program Microsoft Visual Studio.
2. Kliknij opcję **Plik**, a następnie opcję **Nowy** , a następnie opcję **Projekt**.
3. W oknie *Tworzenie nowego projektu*wybierz opcję `IBM MQ .NET Client App (.NET Core)` , a następnie kliknij przycisk **Dalej**.
4. W oknie *Konfigurowanie nowego projektu* zmień wartość w polu *Nazwa projektu* projektu, jeśli chcesz, a następnie kliknij przycisk **Utwórz** , aby utworzyć projekt .NET .

   `MQDotnetApp.cs` jest plikiem tworzonym razem z plikiem projektu. Ten plik zawiera kod, który łączy się z menedżerem kolejek i wykonuje operację put i get.

   Właściwości połączenia są ustawione na wartości domyślne:

   - MQC.CONNECTION_NAME_PROPERTY jest ustawiona na wartość *localhost (1414)*
   - MQC.CHANNEL_PROPERTY jest ustawiony na wartość *DOTNET.SVRCONN*

Kolejka jest ustawiona na *Q1*, a użytkownik może odpowiednio zmodyfikować te właściwości.

5. Skompiluj i uruchom aplikację.

Pojęcia pokrewne

Komponenty i opcje produktu IBM MQ

Odsyłacze pokrewne

Środowisko wykonawcze aplikacji .NET -tylko Windows

Pliki konfiguracyjne dla produktu IBM MQ classes for .NET

Aplikacja kliencka .NET może używać pliku konfiguracyjnego IBM MQ MQI client , a jeśli używany jest typ połączenia zarządzanego, plik konfiguracyjny 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 w przypadku innych IBM MQ MQI client. Plik ten jest zwykle nazywany `mqclient.ini`, ale można podać inną nazwę pliku. Więcej informacji na temat pliku konfiguracyjnego klienta zawiera sekcja Konfigurowanie klienta przy użyciu pliku konfiguracyjnego.

Tylko następujące atrybuty w pliku konfiguracyjnym IBM MQ MQI client mają znaczenie dla produktu IBM MQ classes for .NET. Jeśli zostaną określone inne atrybuty, nie będzie to miało żadnego wpływu.

Tabela 75. Atrybuty pliku konfiguracyjnego klienta, które są istotne dla produktu IBM MQ classes for .NET

sekcja	**Atrybut**
Kanały	CCSID
Kanały	Katalog ChannelDefinition
Kanały	Plik ChannelDefinition
Kanały	ReconDelay
Kanały	DefRecon
Kanały	MQReconnectTimeout
Kanały	Parametry ServerConnection
Kanały	Put1DefaultAlwaysSync
Kanały	PasswordProtection
ŚcieżkaClientExit	ExitsDefaultPath
ŚcieżkaClientExit	ExitsDefaultPath64
MessageBuffer	MaximumSize
MessageBuffer	PurgeTime
MessageBuffer	UpdatePercentage
TCP	ClntRcvBufSize
TCP	ClntSndBufSize
TCP	IPAddressVersion
TCP	KeepAlive

Dowolne z tych atrybutów można przesłonić, używając odpowiedniej zmiennej środowiskowej.

Plik konfiguracyjny aplikacji

Jeśli używany jest typ połączenia zarządzanego, można także przesłonić plik konfiguracyjny klienta IBM MQ i równoważne zmienne środowiskowe za pomocą pliku konfiguracyjnego aplikacji .NET .

Ustawienia pliku konfiguracyjnego aplikacji .NET są wykonywane tylko podczas pracy z typem połączenia zarządzanego i są ignorowane w przypadku innych typów połączeń.

Plik konfiguracyjny aplikacji .NET i jego format są definiowane przez program Microsoft do ogólnego użytku w środowisku produktu .NET , ale poszczególne nazwy sekcji, klucze i wartości wymienione w tej dokumentacji są specyficzne dla produktu IBM MQ.

Format pliku konfiguracyjnego aplikacji .NET jest następujący: *sekcje*. Każda sekcja zawiera jeden lub więcej *kluczy*, a każdy klucz ma powiązaną *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 sekcji i kluczach w sekcji pliku konfiguracyjnego aplikacji .NET są dokładnie zgodne z 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 produktu IBM MQ

Przed użyciem połączenia klienta w celu nawiązania połączenia z menedżerem kolejek należy skonfigurować środowisko produktu IBM MQ .

Uwaga: Ten krok nie jest wymagany, jeśli używany jest produkt IBM MQ classes for .NET w trybie powiązań serwera.

Interfejs programistyczny .NET umożliwia użycie wartości dostosowania NMQ_MQ_LIB, ale obejmuje również klasę MQEnvironment. Ta klasa umożliwia określenie szczegółów, które mają być używane podczas próby nawiązania połączenia, na przykład tych z poniższej listy:

- Nazwa kanału
- Nazwa hosta
- Numer portu
- Wyjścia kanału
- Parametry SSL
- Nazwa i hasło użytkownika

Pełne informacje na temat klasy MQEnvironment można znaleźć w sekcji MQEnvironment.NET class .

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, należy użyć kodu:

```
MQEnvironment.Port = nnnn;
```

Nawiąże połączenie z menedżerem kolejek i odłączenie go od niego

Po skonfigurowaniu środowiska IBM MQ można przystąpić do połączenia 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, wywołaj metodę `Disconnect` w menedżerze kolejek:

```
queueManager.Disconnect();
```

Podczas próby nawiązania połączenia z menedżerem kolejek konieczne jest sprawdzenie uprawnień (`inq`) w menedżerze kolejek. Próba nawiązania połączenia nie powiedzie się bez uzyskiwania informacji o uprawnieniach.

Jeśli zostanie wywołana metoda `Disconnect` , wszystkie otwarte kolejki i procesy, do których użytkownik uzyskuje dostęp za pośrednictwem tego menedżera kolejek, są zamykane. Jednak dobrym rozwiązaniem programowym jest zamknięcie tych zasobów w sposób jawny po zakończeniu korzystania z nich. Aby zamknąć zasoby, należy użyć metody `Close` w obiekcie powiązanym z każdym zasobem.

Metody `Commit` i `Backout` w menedżerze kolejek zastępują wywołania MQCMIT i MQBACK, które są używane z interfejsem proceduralnym.

Dostęp do kolejek i tematów

Dostęp do kolejek i tematów można uzyskać za pomocą metod MQQueueManager lub odpowiednich konstruktorów.

Aby uzyskać dostęp do kolejek, należy użyć metod klasy MQQueueManager . Tabela MQOD (struktura deskryptora obiektu) jest zwinięta do parametrów tych metod. Na przykład, aby 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 należy użyć metody Close (), aby ją zamknąć, tak jak w następującym przykładzie:

```
queue.Close();
```

Za pomocą programu IBM MQ .NETmożna również utworzyć kolejkę przy użyciu konstruktora MQQueue. Parametry są dokładnie takie same, jak w przypadku metody accessQueue , wraz z dodaniem parametru menedżera kolejek określającego instancję obiektu MQQueueManager , która ma być używana. Na przykład:

```
MQQueue queue = new MQQueue(queueManager,
                            "qName",
                            MQC.MQOO_OUTPUT,
                            "qMgrName",
                            "dynamicQName",
                            "altUserId");
```

Konstruowanie obiektu kolejki w ten sposób umożliwia zapisywanie własnych podklas kolejki MQQueue.

Podobnie można również uzyskać dostęp do tematów przy użyciu metod klasy MQQueueManager . Aby otworzyć temat, należy użyć metody AccessTopic(). Zwraca nowy obiekt klasy MQTopic. Po zakończeniu korzystania z tematu należy użyć metody Close () w MQTopic, aby ją zamknąć.

Istnieje również możliwość utworzenia tematu przy użyciu konstruktora MQTopic. Istnieje pewna liczba konstruktorów dla tematów; więcej informacji na ten temat zawiera sekcja Klasa MQTopic.NET.

Obsługa komunikatów

Komunikaty są obsługiwane przy użyciu metod z klas kolejek lub tematów. 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 tabele MQPUT i MQGET są wstawiane i otrzymane tablicami bajtów, produkt IBM MQ classes for .NET wstaw i pobiera instancje klasy MQMessage. Klasa MQMessage hermetykuje bufor danych, który zawiera rzeczywiste dane komunikatu, wraz ze wszystkimi parametrami MQMD (deskryptor komunikatu) opisujących ten komunikat.

Aby zbudować nowy komunikat, należy utworzyć nową instancję klasy MQMessage i użyć metod WriteXXX w celu umieszczenia danych w buforze komunikatów.

Po utworzeniu nowej instancji komunikatu wszystkie parametry MQMD są automatycznie ustawiane na wartości domyślne, zgodnie z definicją w sekcji Wartości początkowe i deklaracje języków dla deskryptora MQMD. Metoda Put () kolejki MQQueue pobiera również instancję klasy opcji MQPutMessagejako parametr. Ta klasa reprezentuje strukturę MQPMO. W poniższym przykładzie tworzony jest 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 () kolejki MQQueue zwraca nową instancję komunikatu MQMessage, która reprezentuje komunikat, który został właśnie odebrany z kolejki. Pobiera ona również instancję klasy opcji MQGetMessagejako parametr. Ta klasa reprezentuje strukturę MQGMO.

Nie ma potrzeby określania maksymalnej wielkości komunikatu, ponieważ metoda Get () automatycznie dostosowuje wielkość swojego wewnętrznego buforu, aby pasował do komunikatu przychodzącego. Aby uzyskać dostęp do danych w zwróconej wiadomości, należy użyć metod ReadXXX klasy MQMessage.

W poniższym przykładzie przedstawiono sposób pobrania 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();
```

Istnieje możliwość zmiany formatu liczb, którego używają metody odczytu i zapisu, ustawiając zmienną elementu *encoding* .

Zestaw znaków używany do odczytu i zapisu 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, a także bajty Unicode, które zawiera. Gdy komunikat zostanie odczytany przez inny program .NET (za pomocą komendy ReadUTF()), jest to najprostszy sposób wysyłania informacji łańcuchowych.

Obsługa właściwości komunikatu

Właściwości komunikatu umożliwiają wybór komunikatów lub pobieranie informacji na temat komunikatu bez uzyskiwania dostępu do jego nagłówków. Klasa MQMessage zawiera metody pobierania i ustawiania właściwości.

Za pomocą właściwości komunikatu można zezwolić aplikacji na wybieranie komunikatów do przetwarzania lub pobieranie informacji o komunikacie bez uzyskiwania dostępu do nagłówków MQMD lub MQRFH2 . Ułatwiają one także komunikację między aplikacjami IBM MQ i JMS . Więcej informacji na temat właściwości komunikatu w produkcie IBM MQzawiera sekcja Właściwości komunikatu.

Klasa MQMessage udostępnia wiele metod pobierania i ustawiania właściwości, zgodnie z typem danych właściwości. Metody get mają nazwy formatu Get * Property, a metody set mają nazwy w formacie Set * Property, gdzie gwiazdka (*) reprezentuje jeden z następujących łańcuchów:

- wartość boolowska
- Byte
- Bajty
- Double
- Float
- Int
- Int2
- Int4
- Int8
- Long
- Obiekt
- Krótki
- Łańcuch

Na przykład, aby uzyskać właściwość myproperty właściwości IBM MQ (łańcuch znaków), należy użyć wywołania `message.GetStringProperty('myproperty')`. Opcjonalnie można przekazać deskryptor właściwości, który IBM MQ zakończy działanie.

Obsługa błędów

Handle errors arising from IBM MQ classes for .NET using `try` and `catch` blocks.

Metody w interfejsie .NET nie zwracają kodu zakończenia i kodu przyczyny. Zamiast tego zgłaszają one wyjątek, gdy kod zakończenia i kod przyczyny wynikające z wywołania IBM MQ nie są jednocześnie zerami. Upraszcza to logikę programu, tak aby nie trzeba było sprawdzać kodów powrotu po każdym wywołaniu programu IBM MQ. Możesz zdecydować, w jakich punktach w programie chcesz poradzić sobie z możliwością awarii. W tych punktach można surround kodu za pomocą bloków `try` i `catch` , jak w następującym 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, które umożliwiają uzyskanie i ustawienie ich wartości atrybutów. Należy zwrócić uwagę, że w przypadku kolejki MQQueue metody działają tylko wtedy, gdy podczas otwierania kolejki użytkownik określi 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 ().

Po utworzeniu nowego obiektu menedżera kolejek przy użyciu operatora *nowy* jest on automatycznie otwierany dla zapytania. Jeśli do uzyskania dostępu do obiektu kolejki używana jest metoda AccessQueue(), obiekt ten nie jest automatycznie otwierany dla operacji sprawdzania lub ustawiania, może to spowodować problemy z niektórymi typami kolejek zdalnych. Aby użyć metod Inquire i Set oraz ustawić właściwości w kolejce, należy określić odpowiednie opcje zapytania i ustawienia w parametrze openOptions metody AccessQueue().

Metody zapytania i ustawiania przyjmują trzy parametry:

- tablica selektorów
- Tablica intAttrs
- Tablica charAttrs

Nie są potrzebne parametry SelectorCount, IntAttrCount i CharAttrLength, które znajdują się w tabeli MQINQ, ponieważ długość tablicy jest zawsze znana. W poniższym przykładzie przedstawiono sposób wykonania 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);
```

Wszystkie atrybuty tych obiektów mogą być zapytania. Podzbiór atrybutów jest prezentowany jako właściwości obiektu. Lista atrybutów obiektów znajduje się w sekcji Atrybuty obiektów. Informacje o właściwościach obiektu można znaleźć w opisie odpowiedniej klasy.

Programy wielowątkowe

Środowisko wykonawcze produktu .NET jest z natury wielowątkowe. Produkt IBM MQ classes for .NET umożliwia współużytkowanie obiektu menedżera kolejek w wielu wątkach, ale zapewnia, że wszystkie uprawnienia dostępu do docelowego menedżera kolejek są zsynchronizowane.

Rozważmy prosty program, który łączy się z menedżerem kolejek i otwiera kolejkę przy starcie. Program wyświetla na ekranie pojedynczy przycisk. Gdy użytkownik kliknie ten przycisk, program pobierze 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 (wątek interfejsu użytkownika).

Implementacja programu IBM MQ .NET zapewnia, że dla konkretnego połączenia (instancji obiektuMQQueueManager) wszystkie uprawnienia dostępu do docelowego menedżera kolejek produktu IBM MQ są zsynchronizowane. Domyślne działanie polega na tym, że wątek, który chce wywołać połączenie z menedżerem 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 współbieżnego dostępu. (Jest to równoznaczne z wywołaniem oddzielnego wywołania MQCONN dla każdego wątku).

Jeśli domyślne opcje połączenia są nadpisywane przez MQC.MQCNO_HANDLE_SHARE_NONE lub MQC.MQCNO_SHARE_NO_BLOCK : menedżer kolejek nie jest już zsynchronizowany.

Korzystanie z tabeli definicji kanału klienta przy użyciu produktu .NET

Za pomocą tabeli definicji kanału klienta (CCDT) można używać klas .NET dla produktu IBM MQ. 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, który nie jest zarządzany przez XA lub XA

W przypadku niezarządzanego typu połączenia można określić położenie tabeli definicji kanału klienta na dwa sposoby:

- Za pomocą zmiennych środowiskowych MQCHLLIB należy określić katalog, w którym znajduje się tabela, oraz wartość MQCHLTAB, aby określić nazwę pliku tabeli.
- Korzystanie z pliku konfiguracyjnego klienta. W sekcji CHANNELS użyj atrybutów ChannelDefinitionDirectory, aby określić katalog, w którym znajduje się tabela, oraz plik ChannelDefinition, aby określić nazwę pliku.

Jeśli położenie jest określone zarówno w pliku konfiguracyjnym klienta, jak i przy użyciu zmiennych środowiskowych, to zmienne środowiskowe mają pierwszeństwo. Za pomocą tej funkcji można określić standardowe położenie w pliku konfiguracyjnym klienta i przesłonić je za pomocą zmiennych środowiskowych, gdy jest to konieczne.

Typ połączenia zarządzanego klienta

W przypadku typu połączenia zarządzanego można określić położenie tabeli definicji kanału klienta na trzy sposoby:

- Korzystanie z pliku konfiguracyjnego aplikacji .NET . W sekcji CHANNELS użyj kluczy ChannelDefinitionDirectory, aby określić katalog, w którym znajduje się tabela, oraz plik ChannelDefinition, aby określić nazwę pliku.
- Za pomocą zmiennych środowiskowych MQCHLLIB należy określić katalog, w którym znajduje się tabela, oraz wartość MQCHLTAB, aby określić nazwę pliku tabeli.
- Korzystanie z pliku konfiguracyjnego klienta. W sekcji CHANNELS użyj atrybutów ChannelDefinitionDirectory, aby określić katalog, w którym znajduje się tabela, oraz plik ChannelDefinition, aby określić nazwę pliku.

Jeśli położenie jest określone w więcej niż jednym z tych sposobów, zmienne środowiskowe mają pierwszeństwo przed plikiem konfiguracyjnym klienta, a plik konfiguracyjny aplikacji .NET ma pierwszeństwo przed obydwoma innymi metodami. Za pomocą tej funkcji można określić standardowe położenie w pliku konfiguracyjnym klienta i przesłonić je za pomocą zmiennych środowiskowych lub pliku konfiguracyjnego aplikacji, gdy jest to konieczne.

Sposób określania definicji kanału, która ma być używana przez aplikację .NET

W środowisku klienta IBM MQ .NET definicja kanału, która ma być używana, może być określona na wiele różnych sposobów. Może istnieć wiele specyfikacji definicji kanału. Aplikacja wywodzi definicję kanału z jednego lub większej liczby źródeł.

Jeśli istnieje więcej niż jedna definicja kanału, to jest on wybierany w następującej kolejności priorytetów:

1. Właściwości określone w konstruktorze MQQueueManager , jawnie lub poprzez włączenie opcji *MQC.CHANNEL_PROPERTY* we właściwościach hashtable
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 (ma zastosowanie tylko do połączeń zarządzanych)
5. Zmienna środowiskowa *MQSERVER*
6. Plik konfiguracyjny klienta, sekcja CHANNELS, atrybut ServerConnectionParms
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 definicji kanału klienta 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, definicja kanału jest wbudowana w pole według wartości z wartości udostępnionych przez aplikację. Te wartości mogą być udostępniane przy użyciu różnych interfejsów i dla każdego z nich może istnieć wiele wartości. Wartości pól są dodawane do definicji kanału zgodnie z podanym porządkiem priorytetowym:

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, cała definicja kanału jest dostarczana jako wartość. Nieokreślone pola w definicji kanału przyjmują wartości domyślne systemowe. Żadne wartości z innych metod definiowania kanałów i ich pól nie są scalane z tymi specyfikacjami.

W przypadku pozycji 7-9 cała definicja kanału jest pobierana z tabeli definicji kanału klienta. Pola, które nie zostały określone jawnie, gdy kanał został zdefiniowany, przyjmują wartości domyślne systemowe. Żadne wartości z innych metod definiowania kanałów i ich pól nie są scalane z tymi specyfikacjami.

Korzystanie z wyjść kanału w programie IBM MQ .NET

Jeśli używane są powiązania klienta, można użyć wyjść kanału, tak jak w przypadku innych połączeń klienckich. Jeśli używane są powiązania zarządzane, należy napisać program obsługi wyjścia, który implementuje odpowiedni interfejs.

Powiązania klienta

Jeśli używane są powiązania klienta, można użyć wyjść kanału zgodnie z opisem w sekcji Wyjścia kanałów. Nie można używać wyjść kanału napisanych dla powiązań zarządzanych.

Powiązania zarządzane

W przypadku korzystania z połączenia zarządzanego w celu zaimplementowania wyjścia należy zdefiniować nową klasę .NET , która implementuje odpowiedni interfejs. W pakiecie IBM MQ zdefiniowane są trzy interfejsy wyjścia:

- MQSendExit
- MQReceiveExit
- MQSecurityExit

Uwaga: Wyjścia użytkownika napisane przy użyciu tych interfejsów nie są obsługiwane jako wyjścia kanału w środowisku niezarządzanym.

Poniższy przykład definiuje klasę, która implementuje wszystkie trzy:

```
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
   }

}
```

Każde wyjście jest przekazywane przez obiekt MQChannelExit i instancję obiektu MQChannelDefinition . Obiekty te reprezentują struktury MQCXP i MQCD zdefiniowane w interfejsie proceduralnym.

Dane, które mają być wysyłane przez wyjście wysyłania, oraz dane odebrane w zabezpieczeniach lub wyjściu odbieranym są określane przy użyciu parametrów wyjścia.

W przypadku pozycji dane w pozycji *dataOffset* o długości *dataLength* w tablicy bajtów *dataBuffer* to dane, które mają zostać wysłane przez wyjście wysyłania, a także dane odebrane w ramach wyjścia bezpieczeństwa lub wyjścia odbierania. Parametr *dataMaxLength* podaje maksymalną długość (z *dataOffset*) dostępny dla wyjścia w *dataBuffer*. Uwaga: W przypadku wyjścia zabezpieczeń dataBuffer może mieć wartość NULL, jeśli jest to pierwsze wywołanie wyjścia lub gdy partner nie został wybrany do wysłania żadnych danych.

W przypadku powrotu wartość *dataOffset* i *dataLength* powinna być ustawiona tak, aby wskazywała przesunięcie i długość w zwróconej tablicy bajtów, która powinna być używana przez klasy produktu .NET . W przypadku wyjścia wysyłania oznacza to, że dane, które powinny zostać wysłane, a także dla wyjścia bezpieczeństwa lub wyjścia odbierania, powinny być interpretowane. Wyjście powinno zwykle zwracać tablicę bajtów; wyjątki to wyjście zabezpieczeń, które może zostać wybrane do wysłania bez danych, a każde wyjście wywołane z przyczyn INIT lub TERM. Najprostszą formą wyjścia, która może zostać zapisana, jest jedna, która nie wykonuje więcej niż zwraca dataBuffer:

Najprostszym możliwym organem wyjściowym jest:

```
{
  return dataBuffer;
}
```

Klasa MQChannelDefinition

Identyfikator użytkownika i hasło, które są określone w aplikacji klienta zarządzanego .NET , są ustawiane w klasie IBM MQ .NET MQChannelDefinition , która jest przekazywana do wyjścia zabezpieczeń klienta. Wyjście zabezpieczeń kopiuje identyfikator użytkownika i hasło do katalogu MQCD.RemoteUserIdentifier i MQCD.RemotePassword (patrz “Zapisywanie wyjścia zabezpieczeń” na stronie 1072).

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, w jakiej są one stosowane, są to:

1. Przekazywanie właściwości hashtable MQC.SECURITY_EXIT_PROPERTY, MQC.SEND_EXIT_PROPERTY lub MQC.RECEIVE_EXIT_PROPERTY w konstruktorze MQQueueManager .
2. Ustawianie właściwości MQEnvironment SecurityExit, SendExit lub ReceiveExit .

Jeśli nazwa kanału i nazwa połączenia nie zostaną określone, wyjście kanału do użycia pochodzi z definicji kanału odezanej z tabeli definicji kanału klienta (CCDT). Nie jest możliwe przesłonięcie wartości zapisanych w definicji kanału. Więcej informacji na temat tabel definicji kanału zawiera sekcja Tabela definicji kanału klienta i “Korzystanie z tabeli definicji kanału klienta przy użyciu produktu .NET” na stronie 599 .

W każdym przypadku specyfikacja przyjmuje formę łańcucha o następującym formacie:

```
Assembly_name(Class_name)
```

nazwa_klasy to pełna nazwa, w tym specyfikacja przestrzeni nazw, klasy .NET , która implementuje IBM.WMQ.MQSecurityExit, IBM.WMQ.MQSendExit lub IBM.WMQ.MQReceiveExit (w zależności od przypadku). *nazwa_zespołu* to pełna nazwa zespołu, w tym rozszerzenie nazwy pliku, w którym znajduje się klasa. Długość łańcucha jest ograniczona do 999 znaków, jeśli używane są właściwości obiektu MQEnvironment lub MQQueueManager. Jeśli jednak nazwa wyjścia kanału jest określona w tabeli definicji kanału klienta, to jest ona ograniczona do 128 znaków. Jeśli jest to konieczne, kod klienta .NET ładuje i tworzy instancję określonej klasy, analizując specyfikację łańcucha.

Określanie danych użytkownika wyjścia kanału (klient zarządzany)

Wyjścia kanału mogą zawierać powiązane z nimi dane użytkownika. Jeśli podczas tworzenia obiektu MQQueueManager zostanie określona nazwa kanału i nazwa połączenia (w środowisku MQEnvironment lub w konstruktorze MQQueueManager), dane użytkownika można określić na dwa sposoby.

W kolejności, w jakiej są one stosowane, są to:

1. Przekazywanie właściwości hashtable MQC.SECURITY_USERDATA_PROPERTY, MQC.SEND_USERDATA_PROPERTY lub MQC.RECEIVE_USERDATA_PROPERTY w konstruktorze MQQueueManager .
2. Ustawianie właściwości danych SecurityUserprogramu MQEnvironment, danych SendUserlub ReceiveUserdanych.

Jeśli nazwa kanału i nazwa połączenia nie zostaną określone, wartości danych użytkownika wyjścia, które mają być używane, pochodzą z definicji kanału odezanej z tabeli definicji kanału klienta (CCDT). Nie jest możliwe przesłonięcie wartości zapisanych w definicji kanału. Więcej informacji na temat tabel definicji kanału zawiera sekcja Tabela definicji kanału klienta i “Korzystanie z tabeli definicji kanału klienta przy użyciu produktu .NET” na stronie 599 .

W każdym przypadku specyfikacja jest łańcuchem, ograniczonym do 32 znaków.

Automatyczne ponowne połączenie klienta w produkcie .NET

Klient może automatycznie ponownie nawiązać połączenie 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 z klientem generowany jest błąd, gdy połączenie nie powiedzie się. Można użyć kodu błędu, aby pomóc w ponownym nawiązaniu połączenia.

Klient korzystający z funkcji automatycznego ponownego połączenia klienta jest nazywany klientem z możliwością ponownego połączenia. Aby utworzyć klienta z możliwością ponownego połączenia, należy określić określone opcje o nazwie opcje ponownego połączenia podczas nawiązywania połączenia z menedżerem kolejek.

Jeśli aplikacja kliencka jest klientem IBM MQ .NET , może on zdecydować się na automatyczne ponowne połączenie klienta, określając odpowiednią wartość dla CONNECT_OPTIONS_PROPERTY, jeśli do utworzenia menedżera kolejek używana jest klasa `MQQueueManager` . Szczegółowe informacje na temat wartości CONNECT_OPTIONS_PROPERTY zawiera sekcja Opcje rekolekcji .

Użytkownik może wybrać, czy aplikacja kliencka zawsze łączy się i ponownie łączy z menedżerem kolejek o tej samej nazwie, z tym samym menedżerem kolejek lub w dowolnym zestawie menedżerów kolejek, które są zdefiniowane z tą samą nazwą `QMNAME` w tabeli połączeń klienta (szczegółowe informacje zawiera sekcja Grupy menedżera kolejek w tabeli CCDT).

Obsługa protokołu TLS (Transport Layer Security) dla produktu .NET

Aplikacje klienckie IBM MQ classes for .NET obsługują szyfrowanie TLS (Transport Layer Security). Protokół TLS zapewnia bezpieczeństwo komunikacji przez internet i umożliwia aplikacjom klienckim/serwerowym komunikowanie się w sposób poufny i niezawodny.

Pojęcia pokrewne

Obsługa protokołu TLS klienta zarządzanego przez program IBM MQ.NET

Protokoły zabezpieczeń szyfrujących: TLS

Obsługa protokołu TLS dla niezarządzanego klienta .NET

Obsługa protokołu TLS dla niezarządzanego klienta .NET jest oparta na bazie danych C MQI i GSKit. Interfejs C MQI obsługuje operacje TLS, a pakiet GSKit implementuje protokoły bezpiecznego gniazda TLS.

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, i musi być zgodny z atrybutem CipherSpec ustawionym na kanale docelowym.

Aby włączyć obsługę protokołu TLS, należy określić wartość parametru CipherSpec przy użyciu statycznej zmiennej składowej SSLCipherSpec produktu MQEnvironment. Poniższy przykład przyłącza się do kanału SVRCONN o nazwie SECURE.SVRCONN.CHANNEL, który został skonfigurowany tak, aby wymagał protokołu TLS z atrybutem CipherSpec o wartości TLS_RSA_WITH_AES_128_CBC_SHA:

```
MQEnvironment.Hostname         = "your_hostname";
MQEnvironment.Channel          = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.SSLCipherSpec    = "TLS_RSA_WITH_AES_128_CBC_SHA256";
MQEnvironment.SSLKeyRepository = "C:\mqm\key";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

Listę CipherSpecszawiera sekcja Określanie specyfikacji CipherSpecs .

Właściwość SSLCipherSpec można także ustawić przy użyciu parametru 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 łańcuchem certyfikatów głównych ośrodka certyfikacji, z którego certyfikat prezentowany przez menedżer kolejek może zostać uwierzytelniony. Podobnie, jeśli parametr SSLClientAuth w kanale SVRCONN został ustawiony na wartość MQSSL_CLIENT_AUTH_REQUIRED, magazyn kluczy klienta musi zawierać identyfikujące certyfikat osobisty, który jest zaufany przez menedżer kolejek.

Korzystanie z nazwy wyróżniającej menedżera kolejek

Menedżer kolejek identyfikuje się za pomocą certyfikatu TLS, który zawiera *nazwę wyróżniającą* (DN).

Aplikacja kliencka IBM MQ .NET może używać tej nazwy wyróżniającej w celu zapewnienia komunikacji z poprawnym menedżerem kolejek. Wzorzec nazwy wyróżniającej jest określany przy użyciu zmiennej nazwy sslPeernazwy środowiska MQEnvironment. Na przykład:

```
MQEnvironment.SSLPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSPHERE";
```

Umożliwia nawiązanie połączenia tylko wtedy, gdy menedżer kolejek wyświetli certyfikat o nazwie Common Name rozpoczynający się od QMGR., i co najmniej dwie nazwy jednostek organizacyjnych, z których pierwszym musi być IBM i drugi WEBSPHERE.

Właściwość SSLPeerName można także ustawić przy użyciu parametru MQC.SSL_PEER_NAME_PROPERTY w tabeli mieszającej właściwości połączenia. Więcej informacji na temat nazw wyróżniających i reguł ustawiania nazw węzłów sieci można znaleźć w sekcji Zabezpieczanie produktu IBM MQ.

Jeśli parametr SSLPeerName jest ustawiony, połączenia powiodą się tylko wtedy, gdy zostanie ustawiony na poprawny wzorzec, a menedżer kolejek wyświetli pasujący certyfikat.

Obsługa błędów podczas używania protokołu TLS

Następujące kody przyczyny mogą być wysyłane przez program IBM MQ classes for .NET podczas nawiązywania połączenia z menedżerem kolejek przy użyciu protokołu TLS:

MQRC_SSL_NOT_ALLOWED
Właściwość SSLCipherSpec została ustawiona, ale używane były połączenia powiązań. Tylko klient łączy obsługę protokołu TLS.

MQRC_SSL_PEER_NAME_MISMATCH
Wzorzec nazwy wyróżniającej określony we właściwości SSLPeerName nie jest zgodny z nazwą wyróżniającą podaną przez menedżer kolejek.

MQRC_SSL_PEER_NAME_ERROR-BŁĄD
Wzorzec nazwy wyróżniającej określony we właściwości SSLPeerName nie jest poprawny.

Obsługa protokołu TLS dla zarządzanego klienta .NET

Zarządzany klient .NET korzysta z bibliotek środowiska Microsoft.NET Framework w celu zaimplementowania protokołów bezpiecznych gniazd TLS. Klasa Microsoft System.Net.SecuritySslStream działa jako strumień przez połączone gniazda TCP, a następnie wysyła i odbiera dane z tego połączenia przez gniazdo.

Minimalny wymagany poziom .NET Framework to .NET Framework v3.5. Poziom obsługi algorytmu Cipher Algorithm jest oparty na poziomie środowiska .NET , który jest używany przez aplikację:

- W przypadku aplikacji opartych na poziomach .NET Framework 3.5 i 4.0dostępne protokoły bezpiecznego gniazda to SSL 3.0 i TSL 1.0.
- W przypadku aplikacji opartych na poziomie .NET Framework 4.5dostępne protokoły bezpiecznego gniazda to SSL 3.0, TLS 1.1 i TLS 1.2.

Może być konieczne przenoszenie aplikacji, które oczekują wyższego obsługi protokołu TLS, do nowszej wersji środowiska zdefiniowanej dla obsługi zabezpieczeń produktu Microsoft w środowisku produktu .NET .

Główne funkcje obsługi protokołu TLS dla zarządzanego klienta .NET są następujące:

Obsługa protokołu TLS
Obsługa protokołu TLS dla klienta zarządzanego .NET jest definiowana za pomocą klasy SSLStream .NET i zależy od środowiska .NET , z którego korzysta aplikacja. Więcej informacji na ten temat zawiera sekcja “Obsługa protokołów TLS dla zarządzanego klienta .NET” na stronie 605.

Obsługa specyfikacji CipherSpec
Ustawienia TLS dla klienta zarządzanego .NET są tak samo jak w przypadku sterów Microsoft.NET TLS. Więcej informacji na ten temat zawierają sekcje “Obsługa specyfikacji szyfrowania CipherSpec dla zarządzanego klienta .NET” na stronie 606 oraz “Odwzorowania CipherSpec dla zarządzanego klienta .NET” na stronie 607.

Repozytoria kluczy
Repozytorium kluczy po stronie klienta to magazyn kluczy produktu Windows . Repozytorium po stronie serwera jest typu repozytorium (CMS) typu Cryptographic Message Syntax (CMS). Więcej informacji na ten temat zawiera sekcja “Repozytoria kluczy dla zarządzanego klienta .NET” na stronie 609.

Certyfikaty
Do zaimplementowania uwierzytelniania wzajemnego 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 609.

SSLPEERNAME
W produkcie .NETaplikacje mogą używać opcjonalnego atrybutu SSLPEERNAME do określenia wzorca nazwy wyróżniającej (Distinguished Name-DN). Więcej informacji na ten temat zawiera sekcja “SSLPEERNAME” na stronie 610.

Zgodność z FIPS
Włączenie trybu FIPS programowo nie jest obsługiwane przez bibliotekę zabezpieczeń Microsoft.NET . Włączenie FIPS jest kontrolowane przez ustawienie strategii grupy produktu Windows .

Zgodność z pakietem NSA Suite B
Produkt IBM MQ implementuje dokument RFC 6460. Implementacja Microsoft.NET dla pakietu NSA Suite B wynosi 5430. Jest to obsługiwane w środowisku .NET Framework 3.5 .

Resetowanie klucza tajnego lub renegocjacja
Chociaż klasa SSLStream nie obsługuje klucza tajnego resetowania lub renegocjacji, dla spójności z innymi klientami IBM MQ , klient zarządzany .NET umożliwia aplikacjom ustawianie liczby SSLKeyReset. Więcej informacji na ten temat zawiera sekcja “Resetowanie lub renegocjowanie klucza tajnego dla zarządzanego klienta .NET” na stronie 610.

Sprawdzanie odwołań
Klasa SSLStream obsługuje sprawdzanie odwołań certyfikatów, które jest automatycznie wykonywane przez mechanizm łączenia certyfikatów. Więcej informacji na ten temat zawiera sekcja “Sprawdzanie odwołań” na stronie 611.

Obsługa wyjścia zabezpieczeń IBM MQ
Klasa SSLStream zapewnia ograniczone wsparcie dla wyjść bezpieczeństwa systemu IBM MQ . Odpytywanie lokalnych i zdalnych certyfikatów w celu uzyskania wartości SSLPeerNamePtr (Subject DN) i SSLRemCertIssNamePtr (Issuer DN) jest możliwe, ponieważ jest to obsługiwane w programie Microsoft.NET. Nie ma jednak obsługi uzyskiwania atrybutów, takich jak DNQ, UNSTRUCTUREDNAME i UNSTRUCTUREDADDRESS, dlatego wartości tych nie można pobrać przy użyciu wyjść.

Obsługa sprzętu szyfrującego
Sprzęt szyfrujący nie jest obsługiwany dla zarządzanego klienta .NET .

Obsługa protokołów TLS dla zarządzanego klienta .NET
IBM MQ.NET TLS support is based on the .NET SSLStream class.

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 604.

W przypadku klasy Microsoft.NET SSLStream w celu zainicjowania protokołu TLS i wykonania uzgadniania za pomocą menedżera kolejek jednym z wymaganych parametrów, które należy ustawić, jest **SSLProtocol**, gdzie należy określić numer wersji TLS, który musi mieć jedną z następujących wartości:

- SSL3.0
- TLS1.0
- TLS1.2

Wartość tego parametru jest ściśle powiązana z rodziną protokołu, do której należy preferowana specyfikacja CipherSpec . Gdy protokół SSLStream uruchamia uzgadnianie TLS z serwerem (menedżerem kolejek), używa on wersji TLS określonej w produkcie **SSLProtocol** do identyfikowania listy specyfikacji CipherSpecs , która ma być używana do negocjacji.

Program IBM MQ.NET nie udostępnia żadnych właściwości, które mogą być używane przez aplikacje do ustawienia tej wartości. Zamiast tego produkt IBM MQ używa tabeli odwzorowania do wewnętrznego odwzorowania zestawu CipherSpec ustawionego na rodzinę protokołów i identyfikuje wersję protokołu SSLProtocol, która ma być używana. W tej tabeli przedstawiono odwzorowanie każdej obsługiwanej wartości CipherSpec między Microsoft.NET i IBM MQoraz wersją protokołu, do której one należą. Więcej informacji na ten temat zawiera sekcja “Odwzorowania CipherSpec dla zarządzanego klienta .NET” na stronie 607.

Obsługa specyfikacji szyfrowania CipherSpec dla zarządzanego klienta .NET
Ustawienia CipherSpec dla aplikacji są używane podczas uzgadniania z serwerem.

Klienty IBM MQ pozwalają na ustawienie wartości CipherSpec używanej podczas uzgadniania z menedżerem kolejek. Klienty IBM MQ powinny ustawić poprawną wartość CipherSpec dla bezpiecznego połączenia w celu ustanowienia, najlepiej CipherSpec określonego w strategii grupy Windows . Pozostawienie tego pola pustego oznacza kanał tekstowy bez żadnych zabezpieczeń w gniazdach.

W przypadku klienta zarządzanego IBM MQ.NET ustawienia TLS są przeznaczone dla klasy SSLStream Microsoft.NET . W przypadku strumienia SSLStream, CipherSpeclub listy preferencji CipherSpecs, można ustawić tylko w strategii grupy Windows , która jest ustawieniem dla całego komputera. Funkcja SSLStream następnie używa podanej listy preferencji CipherSpec lub listy preferencji podczas uzgadniania z serwerem. W przypadku innych klientów IBM MQ właściwość CipherSpec może zostać ustawiona w aplikacji w definicji kanału produktu IBM MQ , a to samo ustawienie jest używane na potrzeby negocjacji TLS. W wyniku tego ograniczenia uzgadnianie TLS może negocjować wszystkie obsługiwane parametry CipherSpec niezależnie od tego, co zostało określone w konfiguracji kanału produktu IBM MQ . Z tego powodu jest prawdopodobne, że spowoduje to błąd AMQ9631 w menedżerze kolejek. Aby uniknąć tego błędu, należy ustawić tę samą wartość parametru CipherSpec jako taką, która została ustawiona w aplikacji jako konfiguracja TLS w strategii grupy Windows .

Nowy kod klienta IBM MQ.NET TLS sprawdza tylko, czy negocjowana była poprawna wersja protokołu. Wersja protokołu TLS jest określana na podstawie specyfikacji CipherSpec , która jest ustawiana przez aplikację i jest używana na potrzeby uzgadniania TLS z serwerem (menedżerem kolejek). W związku z tym projekt CipherSpec w aplikacji klienta zarządzanego IBM MQ.NET musi zostać ustawiony w taki sposób, aby go ustawić. Jeśli właściwość CipherSpec ustawiona przez klient IBM MQ jest inna niż ta z protokołów SSL 3.0, TLS 1.0 i TLS 1.2 , zarządzany klient .NET IBM MQ będzie negocjować domyślnie z dowolnym z szyfrów z protokołów SSL 3.0 lub TLS 1.0 i nie zgłosi błędu.

Uwaga: If the CipherSpec value supplied by the application is not a CipherSpec known to IBM MQ, then the IBM MQ managed .NET client disregards it and negotiates the connection based on the Windows system's group policy.

Ustawianie specyfikacji CipherSpec

Istnieją trzy sposoby ustawiania właściwości CipherSpec:

Klasa MQEnvironment .NET
W poniższym przykładzie przedstawiono sposób ustawienia parametru CipherSpec w klasie 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 TLS CipherSpec
W poniższym przykładzie przedstawiono sposób ustawienia parametru 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 właściwość CipherSpec jest ustawiona w strategii grupy Windows , dla wartości właściwości SSLCipherSpec w kanale SVRCONN i w aplikacji musi być ustawiona taka sama specyfikacja CipherSpec . Jeśli strategia grupy Windows jest ustawiona na wartość domyślną, to znaczy, że strategia grupy nie jest włączona/edytowana dla ustawienia CipherSpec , aplikacje muszą ustawić tę samą wartość domyślną CipherSpec z konfiguracji TLS strategii grupy Windows w klasie MQEnvironment lub w konstruktorze MQQueueManager , które mają właściwości mieszające klasy.

Użycie środowiska CCDT

Program IBM MQ.NET obsługuje tylko tabele definicji kanału klienta (pliki .TAB), które znajdują się na komputerze lokalnym. Istniejące pliki CCDT, które mają ustawioną wartość CipherSpec , mogą być używane dla połączeń IBM MQ.NET . Jednak wartość CipherSpec ustawiona w kanale połączenia klienckiego określa wersję protokołu TLS, a także musi być zgodna z wartością CipherSpec ustawioną w strategii grupy Windows .

Pojęcia pokrewne

“Konfigurowanie środowiska produktu IBM MQ” na stronie 594
Przed użyciem połączenia klienta w celu nawiązania połączenia z menedżerem kolejek należy skonfigurować środowisko produktu IBM MQ .

Zadania pokrewne

Określanie parametru CipherSpecs

Odsyłacze pokrewne

Klasa MQEnvironment .NET

Odwzorowania CipherSpec dla zarządzanego klienta .NET

The IBM MQ.NET interface maintains an IBM MQ to Microsoft.NET mapping table that is used to determine the version of the TLS protocol that the managed client needs to use to establish a secure connection with a queue manager.

Jeśli atrybut CipherSpec jest określony w kanale SVRCONN, to po zakończeniu uzgadniania TLS menedżer kolejek próbuje dopasować tę wartość atrybutu CipherSpec do wynegocjowanej specyfikacji CipherSpec , która jest używana przez aplikację kliencką. Jeśli menedżer kolejek nie może znaleźć zgodnego obiektu CipherSpec, komunikacja nie powiedzie się i wystąpi błąd AMQ9631.

The IBM MQ.NET interface maintains an IBM MQ to Microsoft.NET CipherSpec mapping table. Ta tabela jest używana do określenia wersji protokołu TLS, która ma być używana przez klienta w celu ustanowienia bezpiecznego połączenia gniazda z menedżerem kolejek. W zależności od wartości parametru SSLCipherSpec , protokół SSLProtocol może być protokołem TLS 1.0lub TLS 1.2, w zależności od używanej wersji środowiska Microsoft.NET Framework.

Upewnij się, że podano poprawną wartość parametru SSLCipherSpec , ponieważ podanie niepoprawnej wartości może spowodować użycie protokołów SSL 3.0 lub TLS 1.0 .

Tabela 76. IBM MQ and Microsoft.NET mapping table

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

Tabela 76. IBM MQ and Microsoft.NET mapping table (kontynuacja)		
IBM MQ CipherSpec	**Microsoft.NET CipherSpec**	**Wersja TLS**
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
V 9.1.0.6 ECDHE_RSA_AES_128_GCM_SHA256	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	TLS 1.2
V 9.1.0.6 ECDHE_RSA_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	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
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_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

Uwagi:

1. Ta specyfikacja CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA jest nieaktualna. Jednak może być ona nadal używana do przesyłania do 32 GB danych, zanim połączenie zostanie zakończone z błędem AMQ9288. Aby uniknąć tego błędu, należy unikać używania potrójnego algorytmu szyfrowania DES lub włączyć resetowanie klucza tajnego podczas korzystania z tej specyfikacji CipherSpec.

Repozytoria kluczy dla zarządzanego klienta .NET
Głównym repozytorium używanym przez zarządzane klienty .NET jest magazyn kluczy produktu Windows . Certyfikaty i klucze prywatne muszą być dostępne w magazynie kluczy użytkownika lub systemu, aby mogły być używane przez aplikację kliencką w celu uzyskania tożsamości i zaufania podczas uzgadniania TLS.

Po stronie klienta

Po stronie klienta w konfiguracji TLS w programie IBM MQ.NET składa się repozytorium kluczy po stronie klienta, certyfikaty klienta i opcje, które są używane przez program użytkowy.

- Repozytorium kluczy po stronie klienta jest zawsze plikiem kluczy produktu Windows . Może to być konto użytkownika lub komputera, na podstawie którego mogą być przechowywane certyfikaty.
- W aplikacji można ustawić jedną z następujących wartości dla repozytorium kluczy:
  - "*USER": program IBM MQ.NET uzyskuje dostęp do bazy certyfikatów bieżącego użytkownika w celu pobrania certyfikatów klienta.
  - "*SYSTEM": program IBM MQ.NET uzyskuje dostęp do lokalnego konta komputera 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: Istnieje możliwość zapisania więcej niż jednego certyfikatu w jednym pliku w następujących formatach:

- Wymiana informacji osobistych-PKCS #12 (.PFX, .P12)
- Standard składni komunikatów szyfrujących-certyfikaty PKCS #7 (.P7B)
- Microsoft Serialized Certificate Store (.SST)

Korzystanie z certyfikatów dla zarządzanego klienta .NET
For client certificates, the IBM MQ managed .NET client accesses the Windows keystore and loads all of the client's certificates that are matched either by certificate label or matched by the string.

When selecting a certificate to use, the IBM MQ managed .NET client always uses the first matching certificate for the SSLStream TLS handshake.

Zgodne certyfikaty według etykiety certyfikatu

If you set the certificate label, the IBM MQ managed .NET client searches the Windows certificate store with the given label name to identify the client certificate. Ładuje wszystkie zgodne certyfikaty i korzysta z pierwszego certyfikatu na liście. Istnieją dwie opcje ustawiania etykiety certyfikatu:

- Etykieta certyfikatu może być ustawiona w klasie MQEnvironment uzyskującą dostęp do etykiety MQEnvironment.CertificateLabel.
- Etykieta certyfikatu może być również ustawiona we właściwościach tabeli mieszającej, dostarczanej jako parametr wejściowy z konstruktorem MQQueueManager , jak pokazano w poniższym przykładzie.

```
Hashtable properties = new Hashtable();
properties.Add("CertificateLabel", "mycert");
```

Nazwa ("CertificateLabel") a w wartości rozróżniana jest wielkość liter.

Zgodne certyfikaty według łańcucha

Jeśli etykieta certyfikatu nie jest ustawiona, certyfikat zgodny z łańcuchem `"ibmwebspheremq"` i bieżącym zalogowanym użytkownikiem (małymi literami) jest wyszukiwany i używany.

Zadania pokrewne

Bezpieczne podłączanie klienta do menedżera kolejek

Odsyłacze pokrewne

Klasa MQEnvironment .NET

SSLPEERNAME

Atrybut SSLPEERNAME służy do sprawdzania nazwy wyróżniającej (DN) certyfikatu z menedżera kolejek węzła sieci.

W programie IBM MQ.NETaplikacje mogą używać wartości SSLPEERNAME w celu określenia wzorca nazwy wyróżniającej, tak jak przedstawiono to 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, klient zarządzany IBM MQ.NET nie wykonuje sprawdzania poprawności certyfikatu zdalnego (serwera), a klient zarządzany akceptuje tylko certyfikat zdalny (/serwer).

Sposób ustawienia parametru SSLPEERNAME zależy od tego, które z oferowanych produktów IBM MQ są używane.

IBM MQ classes for .NET

Poniżej przedstawiono trzy opcje.

1. Ustaw właściwość 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 wartość SSLPEERNAME w `Hashtable properties` , jak dla opcji 2.

XMS .NET

Ustaw nazwę węzła sieci SSL w fabryce połączeń:

```
ConnectionFactory.SetStringProperty(XMSC.WMQ_SSL_PEER_NAME, value);
```

WCF

W identyfikatorze URI należy podać nazwę SslPeerName jako rozdzielaną średnikami.

Odsyłacze pokrewne

Klasa MQEnvironment .NET

Resetowanie lub renegocjowanie klucza tajnego dla zarządzanego klienta .NET

Klasa SSLStream nie obsługuje resetowania/renegocjacji klucza tajnego. Jednak, aby zachować spójność z innymi klientami IBM MQ , IBM MQ zarządzany .NET klient umożliwia aplikacjom ustawianie liczby SSLKeyReset.

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

Sprawdzanie odwołań
Klasa SSLStream obsługuje sprawdzanie odwołań certyfikatów.

Sprawdzanie odwołań jest automatycznie wykonywane przez mechanizm łączenia certyfikatów. Dotyczy to zarówno list protokołu OCSP (Online Certificate Status Protocol), jak i list odwołań certyfikatów (Certificate Revocation List-CRL). Klasa SSLStream korzysta z odwołania certyfikatu, które korzysta tylko z serwera określonego w certyfikacie, to znaczy serwer jest podyktowany certyfikatem. Możliwe jest, aby rozszerzenia protokołu HTTP CDP i protokołu OCSP HTTP żądały proxy za pośrednictwem serwera proxy HTTP.

Sposób ustawienia sprawdzania odwołań zależy od tego, które z oferowanych produktów IBM MQ są dostępne.

IBM MQ.NET
Sprawdzenie odwołań można ustawić, uzyskując dostęp do właściwości **MQEnvironment.SSLCertRevocationCheck** w pliku klasy `MQEnvironment.cs` .

XMS .NET
Sprawdzanie odwołań można ustawić w kontekście właściwości fabryki połączeń, tak jak przedstawiono to w poniższym przykładzie.

```
ConnectionFactory.SetBooleanProperty(XMSC.WMQ_SSL_CERT_REVOCATION_CHECK, true);
```

WCF
Sprawdzanie odwołań może być ustawione w identyfikatorze URI zgodnie z następującą konwencją nazewnictwa.

```
"SslCertRevocationCheck=true"
```

Konfigurowanie protokołu TLS na potrzeby zarządzania IBM MQ .NET
Konfigurowanie protokołu TLS na potrzeby zarządzania IBM MQ .NET polega na utworzeniu certyfikatów osoby podpisującej, a następnie konfigurowaniu 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 samopodpisanymi lub certyfikatami udostępnianymi przez ośrodek certyfikacji. Mimo że certyfikaty samopodpisane mogą być używane w systemie programistycznym, testowym lub przedprodukcyjnym, nie należy używać ich w systemie produkcyjnym. W systemie produkcyjnym użyj certyfikatów, które uzyskałeś z zaufanego zewnętrznego ośrodka certyfikacji (CA).

Procedura

1. Utwórz certyfikaty osoby podpisującej.
   a) Aby utworzyć certyfikaty samopodpisane, należy użyć jednego z następujących narzędzi dostępnych w produkcie IBM MQ :

      Z poziomu wiersza komend należy użyć interfejsu GUI produktu **strmqikm** albo produktu **runmqckm** lub **runmqakm** . Więcej informacji na temat korzystania z tych narzędzi zawiera sekcja Korzystanie z produktów **runmqckm**, **runmqakm**i **strmqikm** w celu 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 stronę serwera.
   a) Skonfiguruj protokół TLS w menedżerze kolejek przy użyciu pakietu GSKit, zgodnie z opisem w sekcji bezpiecznie nawiązywanie połączenia z klientem z menedżerem kolejek.

b) Ustaw atrybuty TLS kanału SVRCONN:

- Ustaw wartość **SSLCAUTH** na "REQUIRED/OPTIONAL".
- Ustaw właściwość **SSLCIPH** na odpowiednią wartość CipherSpec.

Więcej informacji na ten temat zawiera sekcja"Włączanie protokołu TLS dla niezarządzanego klienta .NET" na stronie 603.

3. Skonfiguruj stronę klienta.
   a) Zaimportuj certyfikaty klienta do bazy certyfikatów produktu Windows (pod kontem użytkownika/komputera).

   Program IBM MQ .NET uzyskuje dostęp do certyfikatów klienta z bazy certyfikatów Windows , dlatego należy zaimportować certyfikaty do bazy certyfikatów produktu Windows w celu nawiązania bezpiecznego połączenia przez gniazdo z produktem IBM MQ . Więcej informacji na temat uzyskiwania dostępu do magazynu kluczy produktu Windows i importowania certyfikatów po stronie klienta zawiera sekcja Importowanie lub eksportowanie certyfikatów i kluczy prywatnych.

   b) Podaj wartość CertificateLabel zgodnie z opisem w sekcji Połączenie klienta z menedżerem kolejek w bezpieczny sposób.

   c) W razie potrzeby zmodyfikuj strategię grupy Windows w taki sposób, aby ustawić właściwość CipherSpec, a następnie, aby aktualizacje strategii grupy Windows były aktywne, zrestartuj komputer.

4. Skonfiguruj program użytkowy.
   a) Ustaw wartość parametru MQEnvironment lub wartość parametru SSLCipherSpec , aby oznaczyć połączenie jako połączenie chronione.

   Podana wartość jest używana do identyfikowania używanego protokołu (TLS). Zestaw CipherSpec powinien być jednym z elementów CipherSpecs obsługiwanej wersji protokołu SSLProtocol, a najlepiej może być taki sam, jak określony w strategii grupy Windows . (obsługiwana wersja protokołu SSLProtocol jest zależna od używanego środowiska .NET . Protokół SSLProtocol może być w wersji TLS 1.0lub TLS 1.2, w zależności od używanej wersji środowiska Microsoft .NET .)

   Uwaga: If the CipherSpec value supplied by the application is not a CipherSpec known to IBM MQ, then the IBM MQ managed .NET client disregards it and negotiates the connection based on the Windows system's group policy.

   b) Ustaw właściwość SSLKeyRepository na wartość "*SYSTEM" lub "*USER".

   c) Opcjonalne: Ustaw wartość SSLPEERNAME na nazwę wyróżniającą (DN) certyfikatu serwera.

   d) Podaj wartość CertificateLabel zgodnie z opisem w sekcji Połączenie klienta z menedżerem kolejek w bezpieczny sposób.

   e) Należy ustawić dodatkowe wymagane parametry opcjonalne, takie jak KeyResetCount, CertificationRevocationCheck, i włączyć FIPS.

Przykłady ustawiania protokołu TLS i repozytorium kluczy TLS

W przypadku podstawowego .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 , tak jak przedstawiono to 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 aplikacjami zarządzanymi protokołem TLS IBM MQ .NET zawiera sekcja“Pisanie prostej aplikacji” na stronie 613.

Odsyłacze pokrewne

Klasa MQEnvironment .NET

Licznik KeyReset(MQLONG)

Standardy FIPS (Federal Information Processing Standards) dla produktu UNIX, Linux, and Windows

Pisanie prostej aplikacji

Tips for writing a simple IBM MQ managed .NET TLS application, including examples for setting the SSL properties for connection factories, creating a queue manager instance, connection, session and destination, and sending a test message.

Zanim rozpoczniesz

Najpierw należy skonfigurować protokół TLS dla IBM MQ.NET zarządzanego zgodnie z opisem w sekcji “Konfigurowanie protokołu TLS na potrzeby zarządzania IBM MQ .NET” na stronie 611.

W przypadku konfiguracji programu użytkowego w podstawowym produkcie .NETnależy ustawić właściwości protokołu SSL przy użyciu klasy MQEnvironment lub przez dostarczenie tabeli mieszającej jako części konstruktora MQQueueManager .

W przypadku konfiguracji programu użytkowego w programie XMS .NETwłaściwości SSL są ustawiane w kontekście właściwości fabryk połączeń.

Procedura

1. Ustaw właściwości SSL dla fabryk połączeń, tak jak przedstawiono to 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, tak jak przedstawiono to 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 sprawdzić, czy połączenie TLS zostało nawiązane i czy działa poprawnie.

Konfigurowanie śledzenia dla strumienia SSLStream
Aby przechwycić zdarzenia śledzenia i komunikaty związane z klasą SSLStream, należy dodać sekcję konfiguracji diagnostyki systemu do pliku konfiguracyjnego aplikacji dla aplikacji.

O tym zadaniu

If you do not add a configuration section for system diagnostics to the application configuration file, the IBM MQ managed .NET client will not capture any events, traces or debugging points relating to TLS and the SSLStream class.

Uwaga: Uruchomienie śledzenia produktu IBM MQ przy użyciu produktu **strmqtrc** nie powoduje przechwycenia wszystkich wymaganych śledzenia TLS.

Procedura

1. Utwórz plik konfiguracyjny aplikacji (App.Config) dla projektu aplikacji.
2. Dodaj sekcję konfiguracji diagnostyki systemu, tak jak przedstawiono to 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` pozycji `add name` musi być jedną z wersji używanego pliku .net `amqmdnet.dll` .

Przykładowe aplikacje do implementowania protokołu TLS w zarządzanym .NET
Sample applications are provided to show the implementation of TLS for managed .NET in IBM MQ classes for .NET, XMS .NET and IBM MQ custom channel for WCF.

W poniższej tabeli przedstawiono położenie przykładowych aplikacji. *MQ_INSTALLATION_PATH* reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Tabela 77. Położenie przykładowych aplikacji na potrzeby implementowania protokołu TLS w zarządzanym .NET

Oferta stosu IBM MQ.NET	**Położenie próbek**
podstawowe.NET	*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\base\SimplePut\SimplePut.cs *MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\base\SimpleGet\SimpleGet.cs
XMS .NET	*MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\xms\simple\wmq\SimpleProducer\SimpleProducer.cs *MQ_INSTALLATION_PATH*\Tools\dotnet\samples\cs\xms\simple\wmq\SimpleConsumer\SimpleConsumer.cs
Kanał niestandardowy produktu 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: See Features that can be used only with the primary installation on Windows for important information.

Można utworzyć komponenty produktu .NET , których instancje są tworzone za każdym razem, gdy komunikat jest odbierany w monitorowanej kolejce, a następnie przetwarza ten komunikat. Program .NET Monitor jest uruchamiany przez komendę **runmqdnm** i zatrzymany za pomocą komendy **endmqdnm** . Szczegółowe informacje na temat tych komend można znaleźć w sekcji runmqdnm i endmqdnm.

Aby użyć programu .NET Monitor, należy napisać komponent, który implementuje 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 musi być zarejestrowany jako RequiresTransaction lub SupportsTransaction. Nie może ona być zarejestrowana jako RequiresNew , ponieważ monitor .NET zainicjował już transakcję.

Komponent odbiera obiekty MQQueueManager, MQQueue i MQMessage z produktu **runmqdnm**. Może on także odbierać łańcuch parametru użytkownika, jeśli został określony, przy użyciu opcji wiersza komend *-u* , gdy uruchomiono komendę runmqdnm . Należy zauważyć, że komponent odbiera treść komunikatu, który dotarł do monitorowanej kolejki w obiekcie MQMessage. Nie ma potrzeby nawiązywania połączenia z menedżerem kolejek, otwierania kolejki lub pobierania samego komunikatu. Komponent musi następnie przetworzyć komunikat jako odpowiedni i zwrócić kontrolę do programu .NET Monitor.

Jeśli komponent został zapisany jako komponent transakcyjny, zarejestruje się w celu zatwierdzenia lub wycofania transakcji przy użyciu narzędzi udostępnianych przez komponent System.EnterpriseServices.ServicedComponent.

Ponieważ komponent odbiera zarówno obiekty MQQueueManager , jak i obiekty MQQueue, a także komunikat, ma pełne informacje kontekstowe dla tego komunikatu i może na przykład otwierać inną kolejkę w tym samym menedżerze kolejek bez konieczności osobnego łączenia się 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

Przykładowe komendy służące do kompilowania aplikacji produktu .NET napisanych w różnych językach.

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Aby zbudować aplikację C# za pomocą programu IBM MQ classes for .NET, użyj 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 aplikację Zarządzaną C++ przy użyciu produktu 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 dostarczonej przez dostawcę języka.

Korzystanie z autonomicznego klienta IBM MQ .NET

Począwszy od wersji IBM MQ 8.0.0 Fix Pack 2klient 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

Windows V 9.1.1 W systemie IBM MQ 9.1.1biblioteka `amqmdnetstd.dll` jest dostępna dla działu wsparcia .NET Standard w systemie Windows (patrz sekcja “Instalowanie produktu IBM MQ classes for .NET Standard” na stronie 568). 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 .

Linux V 9.1.2 W systemie IBM MQ 9.1.2biblioteka `amqmdnetstd.dll` jest również dostępna w systemie Linux.

O tym zadaniu

Z poziomu produktu IBM MQ 8.0.0 Fix Pack 2można budować aplikacje produktu IBM MQ .NET na komputerze, na którym jest zainstalowany pełny klient IBM MQ , a następnie można spakować zespół produktu IBM MQ .NET (`amqmdnet.dll`) razem z aplikacją i wdrożyć go w systemach produkcyjnych.

Tworzone i wdrażane aplikacje mogą być tradycyjnymi aplikacjami Windows .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 `amqmdnet.dll` w aplikacji w taki sam sposób, jak w przypadku wcześniejszych wersji.

  Ustaw właściwość **CopyLocal** zespołu `amqmdnet` na wartość `True` , aby upewnić się, że zespół `amqmdnet` 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 przy użyciu pliku konfiguracyjnego aplikacji

- Aby uruchomić i zatrzymać śledzenie, należy użyć pliku konfiguracyjnego aplikacji i pliku konfiguracyjnego śledzenia specyficznego dla IBM MQ .

  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:

  – Poniższe kroki dotyczące generowania danych śledzenia dotyczą zarówno klienta zarządzanego podlegającego redystrybucji .NET , jak i autonomicznego klienta .NET .

  – V 9.1.1 Plik konfiguracyjny aplikacji nie jest obsługiwany w systemie .NET Standard. Aby włączyć śledzenie w systemie IBM MQ .NET Standard, należy użyć zmiennej środowiskowej **MQDOTNET_TRACE_ON** .

  Plik konfiguracyjny aplikacji (`app.config` lub `web.config`)

  Aplikacje muszą zdefiniować właściwość **MQTRACECONFIGFILEPATH** w sekcji `<appSettings>` pliku konfiguracyjnego aplikacji, czyli pliku `app.config` lub `web.config` . (Rzeczywista nazwa pliku konfiguracyjnego aplikacji zależy od nazwy aplikacji). Wartość właściwości **MQTRACECONFIGFILEPATH** określa ścieżkę do położenia pliku konfiguracyjnego śledzenia specyficznego dla produktu IBM MQ , `mqtrace.config`, jak pokazano w poniższym przykładzie:

  ```
  <appSettings>
  <add key="MQTRACECONFIGFILEPATH" value="C:\MQTRACECONFIG" />
  </appSettings>
  ```

  Śledzenie jest wyłączone, jeśli plik `mqtrace.config` nie zostanie znaleziony w ścieżce określonej w pliku konfiguracyjnym aplikacji. Jeśli jednak aplikacja ma uprawnienia do zapisu w katalogu bieżącym, dzienniki First Failure Support Technology (FFST) i dzienniki błędów są tworzone w katalogu aplikacji.

  Plik konfiguracyjny śledzenia specyficzny dla systemu IBM MQ (`mqtrace.config`)

  Plik `mqtrace.config` jest plikiem XML, który definiuje właściwości służące do uruchamiania i zatrzymywania śledzenia, ścieżkę do plików śledzenia oraz ścieżkę do dzienników błędów. W poniższej tabeli opisano te właściwości.

Tabela 78. Właściwości zdefiniowane w pliku mqtrace.config	
Atrybut	**Opis**
MQTRACELEVEL	0: zatrzymuje śledzenie-jest to wartość domyślna. 1: uruchamia śledzenie z mniejszymi szczegółami. 2: uruchamia śledzenie z pełnymi szczegółami-zalecane.
MQTRACEPATH	Wskazuje folder, w którym zostaną utworzone pliki śledzenia. Bieżący katalog aplikacji jest używany, jeśli ścieżka jest pusta lub atrybut **MQTRACEPATH** nie jest zdefiniowany.
MQERRORPATH	Wskazuje folder, w którym zostaną utworzone pliki dziennika błędów. Bieżący katalog aplikacji jest używany, jeśli ścieżka jest pusta lub atrybut **MQERRORPATH** nie jest zdefiniowany.

W poniższym przykładzie przedstawiono przykładowy plik `mqtrace.config` :

```
<?xml version="1.0" encoding="utf-8"?>
<traceSettings>
  <MQTRACELEVEL>2</MQTRACELEVEL>
  <MQTRACEPATH>C:\MQTRACEPATH</MQTRACEPATH>
  <MQERRORPATH>C:\MQERRORLOGPATH</MQERRORPATH>
</traceSettings>
```

Śledzenie można uruchamiać i zatrzymywać dynamicznie, gdy aplikacja jest uruchomiona, zmieniając wartość atrybutu **MQTRACELEVEL** w pliku `mqtrace.config` .

Działająca aplikacja musi mieć uprawnienia do tworzenia i zapisu w folderze określonym przez atrybut **MQTRACELEVEL** na potrzeby generowania plików śledzenia. Aplikacje uruchomione w środowisku Microsoft Azure PaaS muszą mieć podobne uprawnienia dostępu, ponieważ aplikacje WWW używające zespołu IBM MQ .NET działającego w środowisku Microsoft Azure PaaS mogą nie mieć uprawnień do tworzenia i zapisu. Generowanie dzienników śledzenia, przechwytywania danych pierwszego niepowodzenia (FDC) i dzienników błędów kończy się niepowodzeniem, jeśli aplikacja nie ma wymaganych uprawnień do tworzenia i zapisu dla określonego folderu.

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

Komponenty i opcje produktu IBM MQ

Klienty podlegające redystrybucji

“Korzystanie z aplikacji WMQDotnetXAMonitor” na stronie 585

Aplikacja WMQDotnetXAMonitor musi być uruchamiana ręcznie. Można ją uruchomić w dowolnym momencie. Można go uruchomić, gdy wyświetlane są komunikaty w systemie SYSTEM.DOTNET.XARECOVERY.QUEUE lub można ją zachować w tle, zanim wykonasz jakiekolwiek prace transakcyjne z aplikacjami napisanych przy użyciu klas IBM MQ .NET .

Odsyłacze pokrewne

Środowisko wykonawcze aplikacji .NET -tylko Windows

Tworzenie aplikacji produktu XMS .NET

IBM Message Service Client for .NET (XMS .NET) udostępnia aplikacyjny interfejs programistyczny (API) o nazwie XMS , który ma ten sam zestaw interfejsów co Java Message Service (JMS) API. Produkt IBM Message Service Client for .NET zawiera w pełni zarządzaną implementację produktu XMS, która może być używana przez dowolny język zgodny z produktem .NET .

O tym zadaniu

Produkt XMS obsługuje:

- Przesyłanie komunikatów w modelu punkt-punkt
- Przesyłanie komunikatów publikowania/subskrypcji
- Synchroniczne dostarczanie komunikatów
- Dostarczanie komunikatów asynchronicznych

Aplikacja XMS może wymieniać komunikaty z następującymi typami aplikacji:

- Aplikacja XMS
- Aplikacja IBM MQ classes for JMS
- Rodzima aplikacja produktu 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 łączyć się z powiązaniami lub trybem klienta.

WebSphere Application Server service integration bus
Aplikacja może korzystać z bezpośredniego połączenia TCP/IP lub może używać protokołu HTTP przez TCP/IP.

IBM Integration Bus
Komunikaty są transportowane między aplikacją a brokerem przy użyciu produktu WebSphere MQ Real-Time Transport. Komunikaty mogą być dostarczane do aplikacji za pomocą programu WebSphere MQ Multicast Transport.

W celu nawiązania połączenia z menedżerem kolejek produktu IBM MQ aplikacja XMS może używać produktu WebSphere MQ Enterprise Transport do komunikowania się z produktem IBM Integration Bus. Alternatywnie aplikacja XMS może publikować i subskrybować połączenie z produktem IBM MQ.

V 9.1.1 From IBM MQ 9.1.1, IBM MQ supports .NET Core for applications in Windows environments. Aby uzyskać więcej informacji, zapoznaj się z sekcją: “Korzystanie z serwera IBM MQ classes for XMS .NET Standard” na stronie 629.

V 9.1.2 From IBM MQ 9.1.2, IBM MQ supports .NET Core for applications in Linux environments.

V 9.1.4 Z poziomu produktu IBM MQ 9.1.4aplikacje zarządzane XMS .NET są w stanie automatycznie zrównoważyć połączenia między menedżerami kolejek klastrowych. Obsługiwane są zarówno biblioteki .NET Framework , jak i .NET Standard . Więcej informacji na ten temat zawiera sekcja Klastry jednolite i Automatyczne równoważenie aplikacji.

Style przesyłania komunikatów obsługiwane przez produkt XMS

Produkt XMS obsługuje style przesyłania komunikatów w trybie punkt z punktem i publikowania/subskrypcji.

Style przesyłania komunikatów są również nazywane domenami przesyłania komunikatów.

Przesyłanie komunikatów w modelu punkt-punkt

Wspólna forma przesyłania komunikatów w trybie punkt z punktem używa kolejkowania. W najprostszym przypadku aplikacja wysyła komunikat do innej aplikacji, identyfikując, niejawnie lub jawnie, kolejkę docelową. Bazowy system przesyłania komunikatów i kolejkowania odbiera komunikat od aplikacji wysyłającej i kieruje do niej komunikat do kolejki docelowej. Aplikacja odbierający może następnie pobrać komunikat z kolejki.

Jeśli bazowy system przesyłania komunikatów i kolejkowania zawiera produkt IBM Integration Bus, produkt IBM Integration Bus może replikować komunikaty i kopie trasy komunikatu do różnych kolejek. W wyniku tego komunikat może otrzymać więcej niż jedna aplikacja. Produkt IBM Integration Bus może także transformować komunikat i dodawać do niego dane.

Kluczowa charakterystyka przesyłania komunikatów w trybie punkt z punktem polega na tym, że aplikacja umieszcza komunikat w kolejce lokalnej, gdy wysyła komunikat. Bazowy system przesyłania komunikatów i kolejkowania określa kolejkę docelową, do której wysyłany jest komunikat. Aplikacja odbierający pobiera komunikat z kolejki docelowej.

Przesyłanie komunikatów publikowania/subskrypcji

W przesyłaniu komunikatów w trybie publikowania/subskrypcji istnieją dwa typy aplikacji: publikator i subskrybent.

Wydawca dostarcza informacje w postaci komunikatów publikacji. Gdy publikator publikuje komunikat, określa on temat, który identyfikuje temat informacji wewnątrz komunikatu.

subskrybent jest konsumentem publikowanych informacji. Subskrybent określa tematy, którymi interesuje się tworzenie subskrypcji.

System publikowania/subskrypcji otrzymuje od subskrybentów publikacje z wydawców i subskrypcje. Kieruje publikacjami do subskrybentów. Subskrybent otrzymuje publikacje dotyczące tylko tych tematów, do których zasubskrybował.

Kluczowa cecha mechanizmu przesyłania komunikatów w trybie publikowania/subskrypcji jest taka, że publikator identyfikuje temat, gdy publikuje komunikat. Nie identyfikuje on subskrybentów. Jeśli komunikat jest publikowany w temacie, dla którego nie ma subskrybentów, to komunikat nie otrzymuje żadnej aplikacji.

Aplikacja może być zarówno publikatorem, jak i subskrybentem.

Model obiektu XMS

Interfejs API produktu XMS jest interfejsem obiektowym. Model obiektu XMS jest oparty na modelu obiektu JMS 1.1 .

Główne klasy produktu XMS

Główne klasy produktu XMS lub typy obiektów są następujące:

ConnectionFactory
Obiekt `ConnectionFactory` hermetyzuje zestaw parametrów dla połączenia. Aplikacja korzysta z `ConnectionFactory` w celu utworzenia połączenia. Aplikacja może udostępniać parametry w czasie wykonywania i tworzyć obiekt `ConnectionFactory` . Alternatywnie parametry połączenia mogą być przechowywane w repozytorium administrowanych obiektów. Aplikacja może pobrać obiekt z repozytorium i utworzyć z niego obiekt `ConnectionFactory` .

Połączenie
Obiekt `Connection` hermetyzuje aktywne połączenie z aplikacji na serwer przesyłania komunikatów. Aplikacja korzysta z połączenia w celu utworzenia sesji.

Miejsce docelowe
Aplikacja wysyła komunikaty lub odbiera komunikaty przy użyciu obiektu `Destination` . W domenie publikowania/subskrybowania obiekt `Destination` obudowuje temat, a w domenie typu punkt z punktem obiekt `Destination` hermetykuje kolejkę. Aplikacja może udostępnić parametry w celu utworzenia obiektu `Destination` w czasie wykonywania. Alternatywnie może on utworzyć obiekt `Destination` na podstawie definicji obiektu przechowywanej w repozytorium obiektów administrowanych.

Sesja
Obiekt `Session` jest jednowątkowym kontekstem do wysyłania i odbierania komunikatów. Aplikacja korzysta z obiektu `Session` do tworzenia obiektów `Message`, `MessageProducer`i `MessageConsumer` .

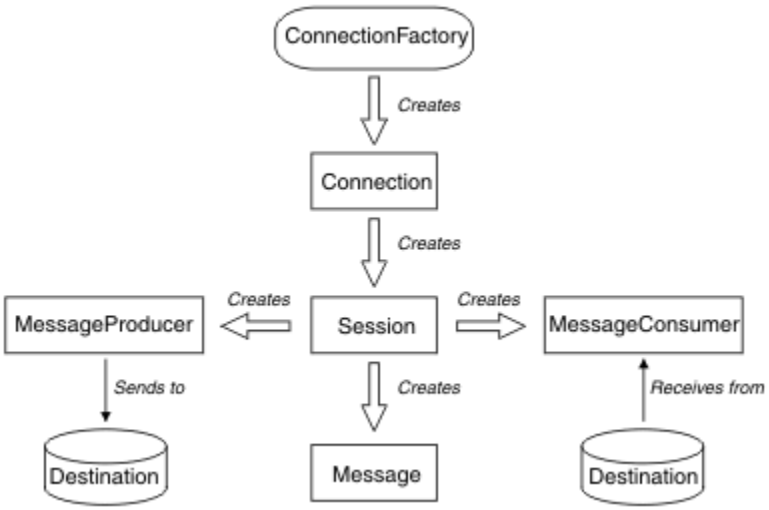
Komunikat
Obiekt `Message` hermetyzuje obiekt `Message` , który aplikacja wysyła za pomocą obiektu `MessageProducer` , lub otrzymuje za pomocą obiektu `MessageConsumer` .

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 58 na stronie 625 przedstawia główne typy obiektów produktu XMS : ConnectionFactory, Connection, Session, MessageProducer, MessageConsumer, Message i Destination. Aplikacja korzysta z fabryki połączeń w celu utworzenia połączenia i korzysta z połączenia w celu utworzenia sesji. Aplikacja może następnie użyć sesji w celu utworzenia komunikatów, producentów komunikatów i konsumentów komunikatów. Aplikacja używa producenta komunikatów do wysyłania komunikatów do miejsca docelowego i używa konsumenta komunikatów do odbierania komunikatów wysyłanych do miejsca docelowego.



Rysunek 58. Obiekty XMS i ich relacje

W programie XMS .NETklasy XMS są definiowane jako zestaw interfejsów produktu .NET . W przypadku kodowania aplikacji produktu XMS .NET wymagane są tylko zadeklarowane interfejsy.

Model obiektów XMS jest oparty na interfejsach niezależnych od domeny, które zostały opisane w specyfikacji Java Message Service , wersja 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 właściwościami obiektu, które są implementowane na różne sposoby:

Atrybuty
Cecha charakterystyczna, która jest zawsze obecna i zajmuje pamięć masową, nawet jeśli atrybut nie ma wartości. W tym zakresie 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 prezentowana i zajmuje pamięć masową dopiero po ustawieniu jej wartości. Nie można usunąć właściwości lub jej pamięć odzyskana po ustawieniu jej wartości. Można zmienić jego wartość. Produkt XMS udostępnia zestaw metod ogólnych do ustawiania i pobierania wartości właściwości.

Administrowane obiekty

Za pomocą administrowanych obiektów można administrować ustawieniami połączenia używalnymi przez aplikacje klienckie, które mają być administrowane z centralnego repozytorium. Aplikacja pobiera definicje obiektów z centralnego repozytorium i używa ich do tworzenia obiektów `ConnectionFactory` i `Destination` . Za pomocą administrowanych obiektów można usunąć kilka aplikacji z zasobów, które są używane w czasie wykonywania.

Na przykład aplikacje produktu XMS mogą być zapisywane i testowane przy użyciu obiektów administrowanych, które odwołują się do zestawu połączeń i miejsc docelowych w środowisku testowym. Po wdrożeniu aplikacji administrowane obiekty można zmienić w celu skonfigurowania aplikacji tak, aby odwoływały się do połączeń i miejsc docelowych w środowisku produkcyjnym.

Produkt XMS obsługuje dwa typy administrowanych obiektów:

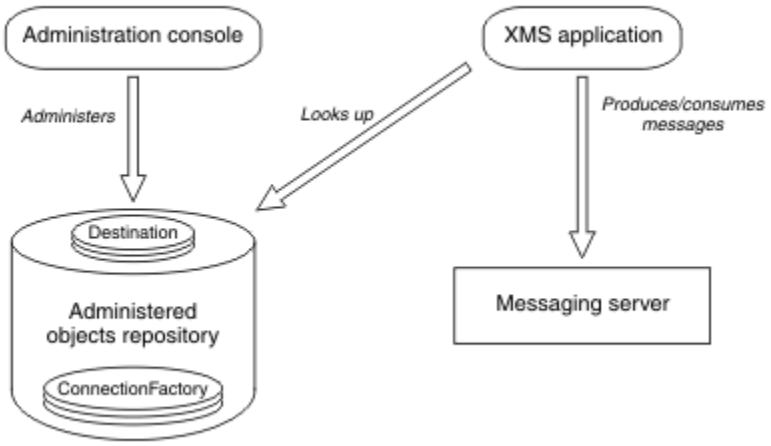
- Obiekt `ConnectionFactory` , który jest używany przez aplikacje do nawiązania początkowego połączenia z serwerem.

- Obiekt `Destination` , który jest używany przez aplikacje do określania miejsca docelowego dla wysyłanych komunikatów oraz do źródła komunikatów odbieranych. Miejsce docelowe jest tematem lub kolejką na serwerze, z którym łączy się aplikacja.

Narzędzie administracyjne **JMSAdmin** jest dostarczane z produktem IBM MQ. Jest on używany do tworzenia obiektów administrowanych i zarządzania nimi w centralnym repozytorium obiektów administrowanych.

Administrowane obiekty w repozytorium mogą być używane przez aplikacje IBM MQ classes for JMS i XMS . Aplikacje produktu XMS mogą używać obiektów `ConnectionFactory` i `Destination` do łączenia się z menedżerem kolejek IBM MQ. Administrator może zmienić definicje obiektów przechowywane w repozytorium bez wpływu na kod aplikacji.

Na poniższym diagramie przedstawiono, w jaki sposób aplikacja XMS zwykle używa administrowanych obiektów. Lewa strona diagramu zawiera repozytorium zawierające definicje obiektów ConnectionFactory i obiektów docelowych, które są administrowane przy użyciu konsoli administracyjnej. Po prawej stronie diagramu wyświetlana jest aplikacja XMS , która wyszukuje definicje obiektów w repozytorium, a następnie korzysta z tych definicji obiektów podczas nawiązywania połączenia z serwerem przesyłania komunikatów.



Rysunek 59. Typowe zastosowanie administrowanych obiektów przez aplikację XMS

Model komunikatów produktu XMS

Model komunikatów produktu XMS jest taki sam, jak model komunikatu produktu IBM MQ classes for JMS .

W szczególności produkt 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 rozpoczynający się przedrostkiem JMS.
- JMS zdefiniowane właściwości. Te pola mają właściwości, których nazwy są rozpoczynanie od przedrostka JMSX.
- IBM zdefiniowane właściwości. Te pola mają właściwości, których nazwy są rozpoczynanie z przedrostkiem JMS_IBM_.

W rezultacie aplikacje produktu XMS mogą wymieniać komunikaty z aplikacjami produktu IBM MQ classes for JMS . W każdym komunikacie niektóre pola nagłówka i właściwości są ustawiane przez aplikację, a inne są ustawiane 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 po jego odebraniu. Pola nagłówka i właściwości są propagowane z komunikatem za pośrednictwem serwera przesyłania komunikatów, gdzie jest to właściwe. Są one dostępne dla każdej aplikacji, która odbiera komunikat.

Pojęcia pokrewne
IBM MQ classes for JMS

Konfigurowanie środowiska serwera przesyłania komunikatów

Tematy w tej sekcji opisują sposób konfigurowania środowiska serwera przesyłania komunikatów w celu umożliwienia aplikacjom produktu XMS nawiązywania połączenia 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 wymagań wstępnych dla aplikacji, które korzystają z połączenia w czasie rzeczywistym z brokerem.

Środowisko serwera przesyłania komunikatów musi zostać skonfigurowane przed uruchomieniem dowolnej aplikacji produktu XMS , w tym przykładowych aplikacji udostępnionych z produktem XMS.

Sekcja obejmuje następujące tematy:

- “Konfigurowanie menedżera kolejek i brokera dla aplikacji, która łączy się z menedżerem kolejek produktu IBM MQ” na stronie 633
- V 9.1.1 “Korzystanie z serwera IBM MQ classes for XMS .NET Standard” na stronie 629
- “Konfigurowanie brokera dla aplikacji, która korzysta z połączenia w czasie rzeczywistym z brokerem” na stronie 634
- “Konfigurowanie magistrali integracji usług dla aplikacji, która łączy się z produktem WebSphere Application Server” na stronie 635

Obiekty nasłuchiwania komunikatów w produkcie XMS .NET

Obiekt nasłuchiwania komunikatów jest używany do asynchronicznego odbierania komunikatu. W przeciwieństwie do wywołania `MessageConsumer.receive()` program nasłuchujący komunikatów nie blokuje wątku wywołującego, zamiast tego dostarcza komunikaty do określonej metody wywołania zwrotnego określonej aplikacji, zwykle jest to metoda **onMessage** .

Dostarczanie komunikatów jest uruchamiane po wywołaniu metody **Connection.Start()** . Dostarczanie komunikatów może zostać zatrzymane i wznowione w dowolnym momencie za pomocą odpowiednio metod **Connection.Stop()** i **Connection.Start()** .

Po wywołaniu metody **Connection.Start()** po ustawieniu obiektu nasłuchiwania komunikatów na co najmniej jednego konsumenta w sesji ta sesja staje się sesją asynchroniczną. Gdy sesja staje się asynchroniczna, nie można wywołać żadnej synchronicznej metody XMS .NET. Na przykład: **MessageProducer.Send()**. Powoduje to wystąpienie wyjątku z kodem przyczyny IBM MQ MQRC_HCONN_ASYNC_ACTIVE (2500).

Wywołania synchroniczne w sesji asynchronicznej

`Session.Close` jest jedynym wywołaniem synchronicznym, który jest dozwolony w sesji asynchronicznej. Aplikacje mogą również wykonywać wywołania synchroniczne (z wyjątkiem `Session.Close`) przy użyciu metody wywołania zwrotnego nasłuchiwania komunikatów, czyli metody **onMessage** .

Oprócz tych dwóch opcji konieczne jest zatrzymanie połączenia przy użyciu metody **Connection.Stop()** dla aplikacji w celu wywołania dowolnego wywołania synchronicznego. Po zakończeniu wywołań konieczne jest ponowne wznowienie połączenia za pomocą metody **Connection.Start()** . który restartuje dostarczanie wiadomości.

Jak wiele asynchronicznych konsumentów komunikatów może mieć sesję?

Sesja może mieć wiele asynchronicznych konsumentów komunikatów. Jednak w każdej chwili komunikat jest dostarczany tylko do jednego konsumenta. W praktyce oznacza to, że po nadejściu drugiego komunikatu, gdy program XMS .NET wywołał metodę **onMessage()** konsumenta w celu dostarczenia pierwszego komunikatu, drugi komunikat nie zostanie dostarczony do konsumenta w sesji, dopóki metoda **onMessage()** nie powróci do systemu.

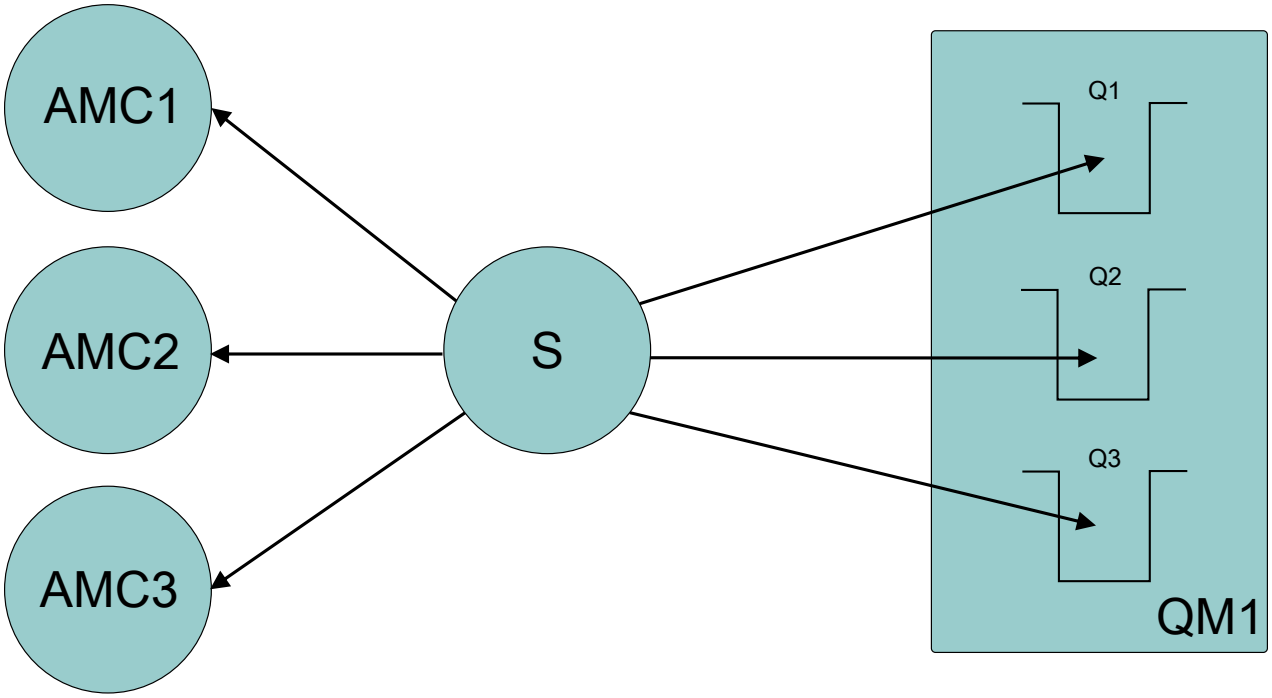
Drugi komunikat jest dostarczany do konsumenta w sesji dopiero po powrocie z metody **onMessage()** . Jest to spowodowane tym, że sesja zarządza dostarczaniem komunikatów do konsumentów przy użyciu tylko jednego wątku. Oznacza to, że tylko jeden komunikat może być dostarczony w danym momencie, a konsument może być dowolny.

Jeśli aplikacja wymaga współbieżnego dostarczania komunikatów, to znaczy, że wszystkie konsumenty muszą odbierać komunikaty w tym samym czasie, aplikacja musi utworzyć wiele sesji, a każda musi mieć jednego konsumenta asynchronicznego komunikatu.

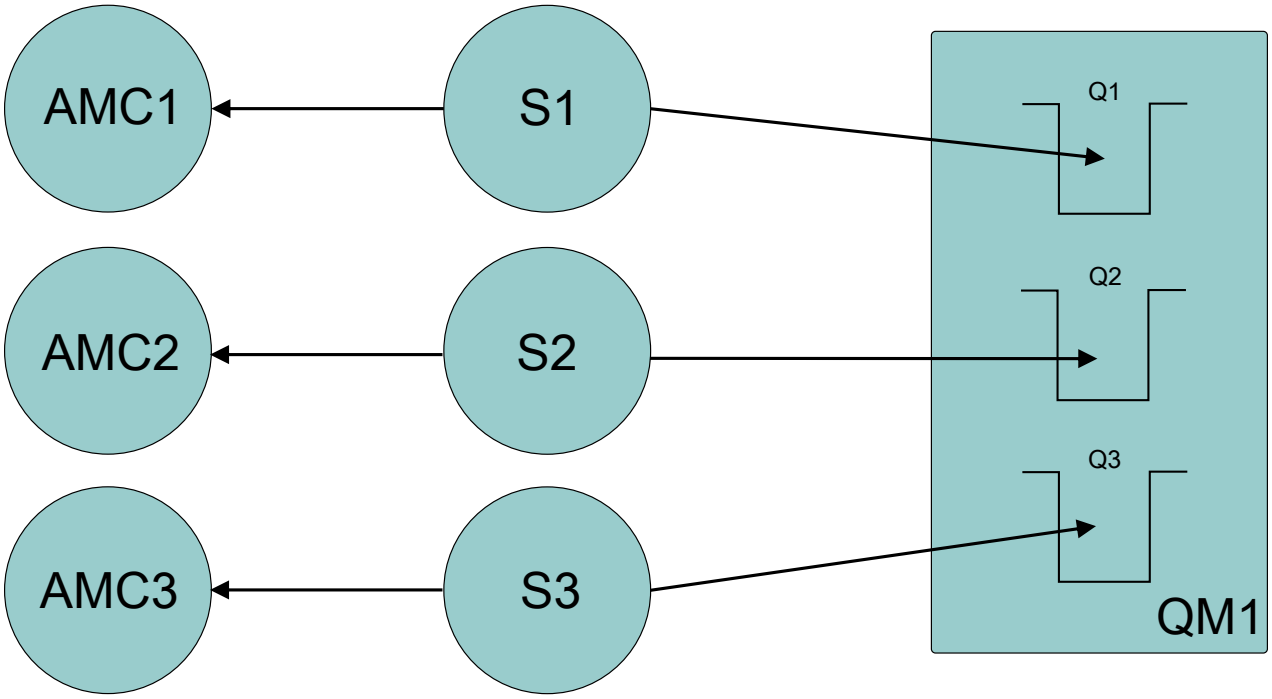
Poniższe przykłady przedstawiają ten składnik w sposób bardziej przejrzysty.

W pierwszym przykładzie w sesji znajduje się wiele asynchronicznych konsumentów komunikatów. Sesja S ma trzy konsumenty komunikatów asynchronicznych: AMC1, AMC2 i AMC3 , które otrzymują komunikaty z trzech różnych miejsc docelowych: Q1, Q2 i Q3.

Ponieważ istnieje tylko jedna sesja S, istnieje tylko wątek dostarczania komunikatów w celu dostarczania komunikatów do konsumentów AMC1, AMC2i AMC3. Gdy sesja dostarcza komunikat do produktu AMC1, pozostałe dwa konsumenty AMC2 i AMC3 czekają, nawet jeśli istnieją komunikaty w produkcie Q2 i Q3 gotowe do dostarczenia.



W drugim przypadku istnieje wiele sesji S1, S2i S3, z których każdy ma jedno asynchroniczne konsumenta komunikatów AMC1, AMC2i AMC3 . Ponieważ dla każdej sesji jest jeden konsument, komunikaty są dostarczane do konsumentów współbieżnie.



To pokazuje, że jeśli wymagane jest współbieżne dostarczanie komunikatów, wymagane jest wiele sesji.

Windows Linux V 9.1.1 Korzystanie z serwera IBM MQ classes for XMS .NET Standard

Sposób użycia języka XMS z produktem Microsoft .NET Standardoraz różnice między używaniem języka IBM MQ classes for XMS .NET Framework i języka IBM MQ classes for XMS .NET Standard. Istnieje wymaganie wstępne produktu Microsoft.NET Core dla systemu IBM MQ classes for XMS .NET Standard.

amqmxmsstd.dll biblioteka

Windows W systemie IBM MQ 9.1.1biblioteka IBM MQ classes for XMS .NET Standard (amqmxmsstd.dll) jest dostępna dla działu wsparcia XMS .NET Standard w systemie Windows.

Linux V 9.1.2 W systemie IBM MQ 9.1.2biblioteka amqmxmsstd.dll jest również dostępna w systemie Linux. Biblioteka jest instalowana w katalogu /&MQINSTALL_PATH&/lib64 path , gdy klient IBM MQ jest instalowany w systemie Linux. XMS Przykłady .NET znajdują się w katalogu &MQINSTALL_PATH&/samp/dotnet/samples/cs/core/xms.

Więcej informacji na ten temat zawiera “Instalowanie produktu IBM MQ classes for .NET Standard” na stronie 568.

Ostrzeżenie: 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 9.1.4 W systemie IBM MQ 9.1.4plik 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 Standard z repozytorium produktu NuGet” na stronie 631.

Komenda dspmqver

W systemie IBM MQ 9.1.1można użyć komendy **dspmqver** , aby wyświetlić informacje o wersji i kompilacji komponentu .NET Core .

Funkcje produktów IBM MQ classes for XMS .NET Framework i IBM MQ classes for XMS .NET Standard

W poniższej tabeli znajduje się lista funkcji produktu IBM MQ 9.1.1, z których można korzystać w produkcie IBM MQ classes for XMS .NET Framework i IBM MQ classes for XMS .NET Standard.

Tabela 79. Różnice w obsłudze funkcji między produktami IBM MQ classes for XMS .NET Framework i IBM MQ classes for XMS .NET Standard

Funkcja	IBM MQ classes for XMS .NET Framework	IBM MQ classes for XMS .NET Standard
Nazwy klas (interfejsy API)	Wszystkie klasy pozostają takie same w każdej sieci.	Wszystkie klasy pozostają takie same w każdej sieci.
System operacyjny	Windows	Windows Kontenery Docker V 9.1.2 Linux V 9.1.3 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 obsługiwany w .NET Standard. Konieczne jest użycie zmiennych środowiskowych w miejsce `app.config`.
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. Pliku `app.config` można użyć do włączenia śledzenia dla pakietu podlegającego redystrybucji.	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

Tabela 79. Różnice w obsłudze funkcji między produktami IBM MQ classes for XMS .NET Framework i IBM MQ classes for XMS .NET Standard (kontynuacja)

Funkcja	IBM MQ classes for XMS .NET Framework	IBM MQ classes for XMS .NET Standard
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. Linux V 9.1.2 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
Obiekty administrowane JNDI	Obsługuje LDAP i FileSystem	Obsługuje tylko system plików FileSystem

Zadania pokrewne

“Korzystanie z przykładowych aplikacji produktu XMS” na stronie 636

Przykładowe aplikacje produktu XMS .NET zawierają przegląd wspólnych funkcji każdego interfejsu API. Można ich używać do weryfikowania konfiguracji instalacji i serwera przesyłania komunikatów oraz do budowania własnych aplikacji.

Windows Linux V 9.1.4 *Pobieranie produktu IBM MQ classes for XMS .NET Standard z repozytorium produktu NuGet*

From IBM MQ 9.1.4, the IBM MQ classes for XMS .NET Standard are available for downloading from the NuGet repository, so that they can be easily consumed by .NET Developers.

O tym zadaniu

NuGet jest menedżerem pakietów dla platform programistycznych Microsoft , w tym .NET. Narzędzia klienckie NuGet umożliwiają tworzenie i korzystanie z pakietów. Pakiet NuGet to pojedynczy plik skompresowany z rozszerzeniem `.nupkg` , który zawiera skompilowany kod (biblioteki DLL), inne pliki związane z tym kodem oraz manifest opisowy zawierający informacje podobne do numeru wersji pakietu.

W programie IBM MQ 9.1.4można pobrać pakiet `IBMXMSDotnetClient` NuGet , który zawiera zarówno bibliotekę `amqmdnetstd.dll` , jak i bibliotekę `amqmxmsstd.dll` , z galerii NuGet , która jest centralnym repozytorium pakietów używanym przez wszystkich autorów i konsumentów pakietów.

Istnieją trzy sposoby pobierania pakietu `IBMXMSDotnetClient` :

- Za pomocą programu Microsoft Visual Studio. Produkt NuGet jest dystrybuowany jako rozszerzenie Microsoft Visual Studio . Z poziomu Microsoft Visual Studio 2012produkt NuGet jest domyślnie instalowany fabrycznie.
- Z poziomu wiersza komend przy użyciu menedżera pakietów produktu NuGet lub interfejsu CLI produktu .NET .
- Za pomocą przeglądarki WWW.

Podobnie jak w przypadku pakietu redystrybucyjnego, śledzenie można włączyć za pomocą zmiennej środowiskowej **XMS_TRACE_ON**.

Procedura

- Aby pobrać pakiet `IBMXMSDotnetClient` przy użyciu interfejsu użytkownika menedżera pakietów w produkcie 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, należy wybrać jedną z następujących opcji:
  - Za pomocą programu 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.
  - Korzystając z interfejsu CLI .NET , wpisz następującą komendę:

    ```
    dotnet add package IBMXMSDotnetClient --version 9.1.4
    ```
- Korzystając z przeglądarki WWW, pobierz pakiet `IBMXMSDotnetClient` z programu https://www.nuget.org/packages/IBMXMSDotnetClient.

Pojęcia pokrewne

"Instalowanie produktu IBM MQ classes for .NET Standard" na stronie 568
Produkty IBM MQ 9.1.1, IBM MQ classes for .NET Standard, w tym przykłady, są instalowane razem z produktem IBM MQ w systemie Windows. V 9.1.2 Z poziomu produktu IBM MQ 9.1.2produkt IBM MQ classes for .NET Standard jest również dostępny na platformach Linux . Istnieje wymaganie wstępne Microsoft.NET Core dla IBM MQ classes for .NET Standard.

Zadania pokrewne

"Pobieranie produktu IBM MQ classes for .NET Standard z repozytorium platformy NuGet" na stronie 571
W serwisie IBM MQ 9.1.4produkt IBM MQ classes for .NET Standard jest dostępny do pobrania z repozytorium NuGet , dzięki czemu może być łatwo używany przez programistów .NET .

Odsyłacze pokrewne
IBM MQ Client for .NET license information

Konfigurowanie menedżera kolejek i brokera dla aplikacji, która łączy się z menedżerem kolejek produktu IBM MQ

W tej sekcji założono, że użytkownik korzysta z produktu IBM WebSphere MQ 7.0.1lub jego nowszej wersji. Zanim możliwe będzie uruchomienie aplikacji łączącej się z menedżerem kolejek produktu IBM MQ , należy skonfigurować menedżer kolejek. W przypadku aplikacji publikowania/subskrypcji dodatkowa konfiguracja jest wymagana, jeśli używany jest interfejs publikowania/subskrybowania w kolejce.

Zanim rozpoczniesz

Produkt XMS działa z produktem IBM Integration Bus lub WebSphere Message Broker 6.1 lub nowszym.

Przed uruchomieniem tego zadania wykonaj następujące kroki:

- Upewnij się, że aplikacja ma dostęp do menedżera kolejek, który jest uruchomiony.
- Jeśli aplikacja jest aplikacją publikowania/subskrypcji i korzysta z interfejsu publikowania/subskrybowania w kolejce, należy upewnić się, że atrybut **PSMODE** jest ustawiony na wartość ENABLED w menedżerze kolejek.
- Upewnij się, że aplikacja korzysta z fabryki połączeń, której właściwości są odpowiednio ustawione w celu nawiązania połączenia z menedżerem kolejek. Jeśli aplikacja jest aplikacją publikowania/subskrypcji, należy upewnić się, że odpowiednie właściwości fabryki połączeń są ustawione na potrzeby używania brokera. Więcej informacji na temat właściwości fabryki połączeń zawiera sekcja Właściwości elementu ConnectionFactory.

O tym zadaniu

Menedżer kolejek i broker można skonfigurować tak, aby uruchamiał aplikacje produktu XMS w taki sam sposób, jak menedżer kolejek i umieszczony w kolejce interfejs publikowania/subskrypcji w celu uruchamiania aplikacji IBM MQ JMS . Poniższe kroki podsumowują to, co należy zrobić.

Procedura

1. W menedżerze kolejek utwórz kolejki, których potrzebuje aplikacja.

   Informacje na temat tworzenia kolejek zawiera sekcja Definiowanie kolejek.

   Jeśli aplikacja jest aplikacją publikowania/subskrypcji i korzysta z interfejsu kolejkowania publikowania/subskrybowania, który wymaga dostępu do kolejek systemowych IBM MQ classes for JMS , przed utworzeniem kolejek poczekaj na krok 4a .
2. Nadaj identyfikatorowi użytkownika powiązanom z aplikacją uprawnienia do łączenia się z menedżerem kolejek, a także odpowiednie uprawnienia dostępu do kolejek.

   Więcej informacji na temat autoryzacji zawiera sekcja Zabezpieczanie. Jeśli aplikacja łączy się z menedżerem kolejek w trybie klienta, patrz także Klienty i serwery.
3. Jeśli aplikacja łączy się z menedżerem kolejek w trybie klienta, należy upewnić się, że kanał połączenia z serwerem jest zdefiniowany w menedżerze kolejek i że obiekt nasłuchiwania został uruchomiony.

   Nie ma potrzeby wykonywania tego kroku dla każdej aplikacji, która łączy się 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 publikowania/subskrypcji i korzysta z interfejsu publikowania/subskrybowania w kolejce, wykonaj następujące kroki.

   a) W menedżerze kolejek utwórz kolejki systemowe produktu IBM MQ classes for JMS , uruchamiając skrypt komend MQSC, które są dostarczane z produktem IBM MQ. Upewnij się, że ID użytkownika powiązany z IBM Integration Bus lub WebSphere Message Broker ma uprawnienia do uzyskiwania dostępu do kolejek.

Więcej informacji na temat miejsca, w którym można znaleźć skrypt i sposób jego uruchomienia, 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 użytkownikowi identyfikator powiązany z aplikacją, aby uzyskać dostęp do kolejek systemowych produktu IBM MQ classes for JMS .

Informacje o tym, jakie uprawnienia są wymagane przez identyfikator 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 komunikaty, które publikują.

Podstawowy przepływ komunikatów składa się z węzła przetwarzania komunikatów MQInput w celu odczytania opublikowanych komunikatów i węzła przetwarzania komunikatów publikowania w celu opublikowania komunikatów.

Informacje na temat sposobu tworzenia i wdrażania przepływu komunikatów można znaleźć w dokumentacji produktu IBM Integration Bus lub WebSphere Message Broker dostępnej w Strona WWW biblioteki dokumentacji produktu IBM Integration Bus.

Nie ma potrzeby wykonywania tego kroku, jeśli odpowiedni przepływ komunikatów jest już wdrożony w brokerze.

Wyniki

Teraz można uruchomić aplikację.

Konfigurowanie brokera dla aplikacji, która korzysta z połączenia w czasie rzeczywistym z brokerem

Przed uruchomieniem aplikacji, która korzysta z połączenia w czasie rzeczywistym z brokerem, należy skonfigurować ten broker.

Zanim rozpoczniesz

Przed rozpoczęciem tej czynności należy wykonać następujące kroki:

- Upewnij się, że aplikacja ma dostęp do brokera, który jest uruchomiony.
- Upewnij się, że aplikacja korzysta z fabryki połączeń, której właściwości są odpowiednio ustawione na potrzeby połączenia w czasie rzeczywistym z brokerem. Więcej informacji na temat właściwości fabryki połączeń zawiera sekcja Właściwości elementu ConnectionFactory.

O tym zadaniu

Istnieje możliwość skonfigurowania brokera do uruchamiania aplikacji produktu XMS w taki sam sposób, jak w przypadku konfigurowania brokera do uruchamiania aplikacji produktu IBM MQ classes for JMS . Poniższe kroki podsumowują to, co należy wykonać:

Procedura

1. Utwórz i wdróż przepływ komunikatów w celu odczytania komunikatów z portu TCP/IP, na którym broker nasłuchuje i opublikuje komunikaty.

   Można to zrobić w jeden z następujących sposobów:

   - Utwórz przepływ komunikatów, który zawiera węzeł przetwarzania komunikatów produktu **RealTimeOptimizedFlow** .

- Utwórz przepływ komunikatów, który zawiera węzeł przetwarzania komunikatów produktu **Real-timeInput** i węzeł przetwarzania komunikatów publikacji.

  Należy skonfigurować węzeł **Real-timeOptimizedFlow** lub **Real-timeInput** , aby nasłuchiwać na porcie używanym do połączeń w czasie rzeczywistym. W programie XMSdomyślnym numerem portu dla połączeń w czasie rzeczywistym jest 1506.

  Nie ma potrzeby wykonywania tego kroku, jeśli odpowiedni przepływ komunikatów jest już wdrożony w brokerze.

2. Jeśli wymagane są komunikaty, które mają zostać dostarczone do aplikacji przy użyciu produktu IBM MQ classes for JMS, należy skonfigurować broker, aby włączyć rozsyłanie grupowe. Należy skonfigurować tematy, które muszą mieć włączone rozsyłanie grupowe, określając niezawodną jakość usługi dla tych tematów, które wymagają niezawodnego rozsyłania grupowego.
3. Jeśli aplikacja udostępnia identyfikator użytkownika i hasło podczas nawiązywania połączenia z brokerem, a użytkownik chce, aby broker uwierzytelnia 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 łączy się z magistralą integracji usług produktu WebSphere Application Server service integration technologies , należy skonfigurować integrację usług w taki sam sposób, w jaki magistrala integracji usług jest skonfigurowana do uruchamiania aplikacji produktu JMS , które korzystają z domyślnego dostawcy przesyłania komunikatów.

Zanim rozpoczniesz

Przed rozpoczęciem tej czynności należy wykonać następujące kroki:

- Upewnij się, że została utworzona magistrala przesyłania komunikatów i że serwer jest dodawany do magistrali jako element magistrali.
- Upewnij się, że aplikacja ma dostęp do magistrali integracji usług, która zawiera co najmniej jeden działający mechanizm przesyłania komunikatów.
- Jeśli wymagane jest wykonanie operacji HTTP, należy zdefiniować kanał transportowy przychodzący mechanizmu przesyłania komunikatów HTTP. Domyślnie kanały dla protokołów SSL i TCP są definiowane podczas instalacji serwera.
- Upewnij się, że aplikacja korzysta z 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, który ma być używany podczas negocjowania połączenia z serwerem przesyłania komunikatów (czyli za pośrednictwem serwera startowego). W najprostszej postaci, dla serwera zainstalowanego z ustawieniami domyślnymi, punkt końcowy udostępniania może być ustawiony na nazwę hosta serwera.
  - Nazwa magistrali, przez którą wysyłane są komunikaty.

  Więcej informacji na temat właściwości fabryki połączeń zawiera sekcja Właściwości elementu ConnectionFactory.

O tym zadaniu

Należy zdefiniować wszystkie wymagane przez użytkownika obszary kolejki lub tematu. Domyślnie obszar tematu o nazwie Default.Topic.Space jest zdefiniowany podczas instalacji serwera, ale jeśli wymagane są dalsze obszary tematów, należy samodzielnie utworzyć te obszary tematów. Nie ma potrzeby wstępnego

definiowania poszczególnych tematów w obrębie obszaru tematu, ponieważ serwer tworzy instancje tych pojedynczych tematów w sposób dynamiczny.

Poniższe kroki podsumowują to, co należy zrobić.

Procedura

1. Utwórz kolejki, które będą potrzebne aplikacji do przesyłania komunikatów w trybie punkt z punktem.
2. Utwórz dodatkowe obszary tematów, które będą potrzebne aplikacji do przesyłania komunikatów w trybie publikowania/subskrypcji.

Wyniki

Teraz można uruchomić aplikację.

Korzystanie z przykładowych aplikacji produktu XMS

Przykładowe aplikacje produktu XMS .NET zawierają przegląd wspólnych funkcji każdego interfejsu API. Można ich używać do weryfikowania konfiguracji instalacji i serwera przesyłania komunikatów oraz do budowania własnych aplikacji.

O tym zadaniu

Jeśli potrzebna jest pomoc w tworzeniu własnych aplikacji, można użyć przykładowych aplikacji jako punktu początkowego. Zarówno źródło, jak i skompilowana wersja są udostępniane dla każdej aplikacji. Zapoznaj się z przykładowym kodem źródłowym i zidentyfikuj kroki kluczowe, aby utworzyć każdy wymagany obiekt dla aplikacji (ConnectionFactory, Connection, Session, Destination, Producer, Consumer lub both), a także aby ustawić wszystkie właściwości wymagane do określenia sposobu działania aplikacji. Więcej informacji na ten temat zawiera “Pisanie aplikacji produktu XMS” na stronie 639. Przykłady mogą ulec zmianie w przyszłych wersjach produktu XMS.

W poniższej tabeli przedstawiono zestawy przykładowych aplikacji (po jednym dla każdego interfejsu API), które są dostarczane razem z produktem XMS.

Tabela 80. Przykładowe aplikacje dla XMS .NET

Nazwa próbki	**Opis**
SampleConsumerCS	Aplikacja konsumująca komunikaty, 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 administrowanego repozytorium obiektów, które jest oparte na plikach. Aplikacja zawiera fabrykę połączeń i miejsce docelowe dla określonych ustawień połączenia. Administrowane repozytorium obiektów może być następnie używane razem z każdym przykładowym konsumentem i aplikacjami producenta.

Przykłady, które obsługują te same funkcje w różnych interfejsach API, mają różnice syntaktyczne.

- Przykładowe aplikacje konsumenta i producenta komunikatów obsługują następujące funkcje:
  - Połączenia z IBM MQ, IBM Integration Bus (przy użyciu połączenia w czasie rzeczywistym z brokerem) oraz WebSphere Application Server service integration bus
  - Wyszukiwanie administrowanych repozytorium obiektów przy użyciu początkowego interfejsu kontekstu

  - Połączenia z kolejkami (IBM MQ i WebSphere Application Server service integration bus) oraz tematy (IBM MQ, połączenie w czasie rzeczywistym z brokerem i WebSphere Application Server service integration bus)
  - Komunikaty podstawowe, bajtowe, mapy, obiekty, strumienie i tekst
- Przykładowa aplikacja konsumująca wiadomości obsługuje synchroniczne i asynchroniczne tryby odbierania oraz instrukcje Selector SQL.
- Przykładowa aplikacja producenta komunikatów obsługuje trwałe i nietrwałe tryby dostarczania.

Próbki mogą działać w jednym z dwóch trybów:

tryb prosty
Użytkownik może uruchomić przykłady z minimalnym wejściem użytkownika.

tryb zaawansowany
Bardziej precyzyjnie dostosować sposób działania przykładów.

Wszystkie próbki są kompatybilne i mogą w związku z tym działać w różnych językach.

Windows From IBM MQ 9.1.1, IBM MQ supports .NET Core for XMS .NET applications in Windows environments. IBM MQ classes for .NET Standard, w tym przykłady, są instalowane domyślnie jako część standardowej instalacji produktu IBM MQ .

Linux V 9.1.2 Z poziomu produktu IBM MQ 9.1.2produkt IBM MQ obsługuje również moduł .NET Core dla aplikacji w środowiskach Linux .

Przykładowe aplikacje dla programu XMS .NET są zainstalowane w produkcie `&MQINSTALL_PATH&/samp/dotnet/samples/cs/core/xms`.

Więcej informacji na ten temat zawiera “Korzystanie z serwera IBM MQ classes for XMS .NET Standard” na stronie 629.

Uruchamianie przykładowych aplikacji produktu .NET

Przykładowe aplikacje produktu .NET można uruchomić interaktywnie w trybie prostym lub zaawansowanym, albo w trybie nieinteraktywnym, używając automatycznie wygenerowanych lub dostosowanych plików odpowiedzi.

Zanim rozpoczniesz

Przed uruchomieniem dowolnej z dostarczonych przykładowych aplikacji należy najpierw skonfigurować środowisko serwera przesyłania komunikatów, aby aplikacje mogły łączyć się z serwerem. Patrz “Konfigurowanie środowiska serwera przesyłania komunikatów” na stronie 627.

Procedura

Aby uruchomić przykładową aplikację .NET, wykonaj następujące kroki:

Wskazówka: Kiedy uruchamiasz przykładową aplikację, wpisz? w każdej chwili, aby uzyskać pomoc na temat tego, co dalej.

1. Wybierz tryb, w którym ma zostać uruchomiona aplikacja przykładowa.

   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 tym przypadku wartością domyślną jest `wpm` (połączenie z serwerem WebSphere Application Server service integration bus).

Wyniki

Po uruchomieniu przykładowych aplikacji pliki odpowiedzi są generowane automatycznie w bieżącym katalogu roboczym. Nazwy plików odpowiedzi są w formacie *connection_type*`-`*sample_type*`.rsp`, na przykład `wpm-producer.rsp`. Jeśli jest to wymagane, można użyć wygenerowanego pliku odpowiedzi, aby ponownie uruchomić aplikację przykładową z tymi samymi opcjami, tak aby nie było konieczne ponowne wprowadzanie opcji.

Zadania pokrewne

Budowanie przykładowych aplikacji produktu .NET
Podczas budowania przykładowej aplikacji produktu .NET tworzona jest wersja wykonywalna wybranej próby.

Budowanie własnych aplikacji
Użytkownik buduje własne aplikacje, takie jak kompilacja przykładowych aplikacji.

Budowanie przykładowych aplikacji produktu .NET

Podczas budowania przykładowej aplikacji produktu .NET tworzona jest wersja wykonywalna wybranej próby.

Zanim rozpoczniesz

Zainstaluj odpowiedni kompilator. W przypadku tego zadania założono, że użytkownik ma zainstalowany produkt Microsoft Visual Studio 2012 i że jest on zaznajomiony z jego używaniem.

Procedura

Aby zbudować przykładową aplikację produktu .NET , wykonaj następujące kroki:

1. Kliknij plik rozwiązania `Samples.sln` dostarczony wraz z przykładami produktu .NET .
2. Kliknij prawym przyciskiem myszy rozwiązanie `Przykłady` w oknie eksploratora rozwiązań i wybierz opcję **Zbuduj 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# aplikacji przykładowej producenta komunikatów, produkt `SampleProducerCS.exe` jest tworzony w folderze `SampleProducer` .

Zadania pokrewne

Uruchamianie przykładowych aplikacji produktu .NET
Przykładowe aplikacje produktu .NET można uruchomić interaktywnie w trybie prostym lub zaawansowanym, albo w trybie nieinteraktywnym, używając automatycznie wygenerowanych lub dostosowanych plików odpowiedzi.

Budowanie własnych aplikacji
Użytkownik buduje własne aplikacje, takie jak kompilacja przykładowych aplikacji.

“Budowanie własnych aplikacji” na stronie 638
Użytkownik buduje własne aplikacje, takie jak kompilacja przykładowych aplikacji.

Budowanie własnych aplikacji

Użytkownik buduje własne aplikacje, takie jak kompilacja przykładowych aplikacji.

Zanim rozpoczniesz

Zainstaluj odpowiedni kompilator. W przypadku tego zadania założono, że użytkownik ma zainstalowany produkt Microsoft Visual Studio 2012 i że jest on zaznajomiony z jego używaniem.

Procedura

- Zbuduj aplikację .NET zgodnie z opisem w sekcji “Budowanie przykładowych aplikacji produktu .NET” na stronie 638.

  Aby uzyskać dodatkowe wskazówki dotyczące sposobu budowania własnych aplikacji, należy użyć plików makefile udostępnionych dla każdej przykładowej aplikacji.

  Wskazówka: Aby ułatwić diagnozowanie problemów w przypadku wystąpienia awarii, pomocne może być skompilowanie aplikacji z dołączonym symbolami.

Zadania pokrewne

Uruchamianie przykładowych aplikacji produktu .NET
Przykładowe aplikacje produktu .NET można uruchomić interaktywnie w trybie prostym lub zaawansowanym, albo w trybie nieinteraktywnym, używając automatycznie wygenerowanych lub dostosowanych plików odpowiedzi.

Budowanie przykładowych aplikacji produktu .NET
Podczas budowania przykładowej aplikacji produktu .NET tworzona jest wersja wykonywalna wybranej próby.

Pisanie aplikacji produktu XMS

Tematy zawarte w tej sekcji zawierają informacje pomocne podczas pisania aplikacji XMS w ogóle.

O tym zadaniu

Ta sekcja zawiera ogólne pojęcia związane z pisaniem aplikacji produktu XMS . Więcej informacji na temat tworzenia aplikacji produktu XMS .NET można znaleźć w sekcji “Pisanie aplikacji produktu XMS .NET” na stronie 659 .

Sekcja obejmuje następujące tematy:

- “Model wielowątkowości” na stronie 640
- “Obiekty ConnectionFactories i obiekty Connection” na stronie 641
- “Sesje” na stronie 642
- “Miejsca docelowe” na stronie 646
- “Producenci komunikatów” na stronie 649
- “Konsumenci komunikatów” na stronie 650
- “Przeglądarki kolejek” na stronie 654
- “Requestery” na stronie 654
- “Usuwanie obiektu” na stronie 655
- “Typy podstawowe produktu XMS” na stronie 656
- “Niejawne przekształcenie wartości właściwości z jednego typu danych na inny.” na stronie 656
- “Iteratory” na stronie 658
- “Identyfikatory kodowanego zestawu znaków” na stronie 659
- “Kody błędów i wyjątków produktu XMS” na stronie 659
- “Budowanie własnych aplikacji” na stronie 638

Windows V 9.1.5 Korzystanie z szablonu projektu IBM MQ XMS .NET

W produkcie IBM MQ 9.1.5klient IBM MQ XMS .NET oferuje możliwość korzystania z szablonu projektu w celu ułatwienia tworzenia aplikacji XMS .NET Core .

Zanim rozpoczniesz

W systemie muszą być dostępne Microsoft Visual Studio 2017(lub nowsze) oraz .NET Core 2.1 (w systemie).

Należy skopiować szablon XMS .NET z

```
&MQ_INSTALL_ROOT&\tools\dotnet\samples\cs\core\xms\ProjectTemplates\IBMXMS.NETClientApp.zip
```

katalog do

```
&USER_HOME_DIRECTORY&\Documents\&Visual_Studio_Version&\Templates\ProjectTemplates
```

Katalog, w którym:

- *& MQ_INSTALL_ROOT* to katalog główny instalacji
- Katalog *& USER_HOME_DIRECTORY* jest katalogiem osobistym użytkownika.

Aby można było odebrać szablon, należy zatrzymać i zrestartować produkt Microsoft Visual Studio .

O tym zadaniu

Szablon projektu produktu XMS .NET zawiera wspólny kod, którego można użyć w celu ułatwienia tworzenia aplikacji. Za pomocą wbudowanego kodu można nawiązać połączenie z menedżerem kolejek produktu IBM MQ , a następnie wykonać operację put lub get, modyfikując po prostu właściwości w kodzie wbudowanym.

Procedura

1. Otwórz program Microsoft Visual Studio.
2. Kliknij opcję **Plik**, a następnie opcję **Nowy** , a następnie opcję **Projekt**.
3. W oknie *Tworzenie nowego projektu*wybierz opcję `IBM XMS .NET Client App (.NET Core)` , a następnie kliknij przycisk **Dalej**.
4. W oknie *Konfigurowanie nowego projektu* zmień wartość w polu *Nazwa projektu* projektu, jeśli chcesz, a następnie kliknij przycisk **Utwórz** , aby utworzyć projekt XMS .NET .

   `XMSDotnetApp.cs` jest plikiem tworzonym razem z plikiem projektu. Ten plik zawiera kod, który łączy się z menedżerem kolejek, a następnie wykonuje operację wysyłania i odbierania.

   Właściwości połączenia są ustawione na wartości domyślne:

   - Wartość WMQ_CONNECTION_NAME_LIST jest ustawiona na *localhost (1414)*
   - XMSC.WMQ_CHANNEL jest ustawiona na wartość *DOTNET.SVRCONN*

   Kolejka jest ustawiona na *Q1*, a użytkownik może 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

Ogólne reguły zarządzają sposobem, w jaki aplikacja wielowątkowa może używać obiektów XMS .

- Tylko obiekty z następujących typów mogą być używane współbieżnie w różnych wątkach:
  - ConnectionFactory
  - Połączenie
  - Dane ConnectionMeta
  - Miejsce docelowe
- Obiekt Session może być używany tylko w jednym wątku w dowolnym momencie.

Wyjątki od tych reguł są oznaczane przez wpisy z etykietą "Kontekst wątku" w definicjach interfejsu metod w IBM Message Service Client for .NET reference.

Obiekty ConnectionFactories i obiekty Connection

Obiekt ConnectionFactory udostępnia szablon, który jest używany przez aplikację do tworzenia obiektu połączenia. Aplikacja korzysta z obiektu połączenia w celu utworzenia obiektu sesji.

W przypadku produktu .NETaplikacja XMS najpierw używa obiektu XMSFactoryFactory do uzyskania odwołania do obiektu ConnectionFactory , który jest odpowiedni do wymaganego typu protokołu. Ten obiekt ConnectionFactory może następnie generować połączenia tylko dla tego typu protokołu.

Aplikacja XMS może tworzyć wiele połączeń, a aplikacja wielowątkowa może korzystać jednocześnie z pojedynczego obiektu połączenia w wielu wątkach. Obiekt połączenia hermetyzuje połączenie komunikacyjne między aplikacją a serwerem przesyłania komunikatów.

Połączenie służy kilku celom:

- Gdy aplikacja tworzy połączenie, aplikacja może zostać uwierzytelniona.
- 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 obiekcie ConnectionFactory specyficznym dla klienta przy użyciu właściwości i w sposób przezroczysty przypisanie go do tworzonego połączenia.

  Alternatywnym sposobem przypisywania identyfikatora klienta jest użycie wartości specyficznej dla dostawcy, która jest ustawiona w obiekcie połączenia. Ta wartość nie przesłania identyfikatora, który został skonfigurowany administracyjnie. Jest on dostępny dla przypadku, w którym nie istnieje określony administracyjny identyfikator. Jeśli podany identyfikator administracyjny nie istnieje, próba przesłonięcia go z wartością specyficzną dla dostawcy powoduje zgłoszenie wyjątku. Jeśli aplikacja jawnie ustawi identyfikator, musi to zrobić natychmiast po utworzeniu połączenia, a przed innymi działaniami w tym połączeniu. W przeciwnym razie zgłaszany jest wyjątek.

Aplikacja XMS zwykle tworzy połączenie, jedną lub więcej sesji oraz liczbę producentów komunikatów i konsumentów komunikatów.

Tworzenie połączenia jest stosunkowo kosztowne pod względem zasobów systemowych, ponieważ wiąże się on z nawiązaniem połączenia komunikacyjnego, a także może wymagać uwierzytelnienia aplikacji.

Uruchomiono i zatrzymano połączenie

Połączenie może działać w trybie uruchomionym lub zatrzymanego.

Gdy aplikacja tworzy połączenie, połączenie jest w trybie zatrzymanego. Gdy połączenie jest w trybie zatrzymanych, aplikacja może inicjować sesje i może wysyłać komunikaty, ale nie może ich odbierać, synchronicznie lub asynchronicznie.

Aplikacja może uruchomić połączenie, wywołując metodę `Start Connection` . Gdy połączenie jest w trybie uruchomionym, aplikacja może wysyłać i odbierać komunikaty. Aplikacja może następnie zatrzymać i zrestartować połączenie, wywołując metody zatrzymania połączenia i `Start Connection` .

Zamknięcie połączenia

Aplikacja zamknie połączenie, wywołując metodę Close Connection (Zamknij połączenie). Gdy aplikacja zamknie połączenie, program XMS wykonuje następujące działania:

- Zamyka ona wszystkie sesje powiązane z połączeniem i usuwa niektóre obiekty powiązane z tymi sesjami. Więcej informacji o tym, które obiekty są usuwane, zawiera sekcja "Usuwanie obiektu" na stronie 655. W tym samym czasie program XMS wycofuje wszystkie transakcje, które są aktualnie w toku w ramach sesji.
- Kończy on połączenie komunikacyjne z serwerem przesyłania komunikatów.
- Zwalnia ona pamięć i inne zasoby wewnętrzne używane przez połączenie.

Program XMS nie potwierdza odbioru żadnych komunikatów, które nie potwierdziły podczas sesji, przed zamknięciem połączenia. Więcej informacji na temat potwierdzania odbioru komunikatów zawiera sekcja "Potwierdzenie komunikatu" na stronie 644.

Obsługa wyjątków

Wyjątki XMS .NET wywodzi się z wyjątku System.Exception. Więcej informacji na ten tematzawiera sekcja "Obsługa błędów w produkcie .NET" na stronie 663.

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 przy użyciu protokołu TCP/IP.

Protokół HTTP może być używany w sytuacjach, gdy bezpośrednie połączenie TCP/IP nie jest możliwe. Jedną z typowych sytuacji jest komunikacja za pośrednictwem firewalla, na przykład w przypadku wymiany komunikatów przez dwa przedsiębiorstwa. Używanie protokołu HTTP do komunikowania się za pośrednictwem firewalla jest często nazywane *tunelowaniem HTTP*. Tunelowanie HTTP jest jednak z natury wolniejsze niż użycie bezpośredniego połączenia TCP/IP, ponieważ nagłówki HTTP dodają znacznie do ilości przesyłanych danych, a protokół HTTP wymaga większej liczby przepływów komunikacji niż TCP/IP.

Aby utworzyć połączenie TCP/IP, aplikacja może używać fabryki połączeń, której właściwość XMSC_WPM_TARGET_TRANSPORT_CHAIN jest ustawiona na wartość XMSC_WPM_TARGET_TRANSPORT_CHAIN_BASIC. Jest to wartość domyślna właściwości. Jeśli połączenie zostanie utworzone pomyślnie, właściwość XMSC_WPM_CONNECTION_PROTOCOL połączenia zostanie ustawiona na wartość XMSC_WPM_CP_TCP.

Aby utworzyć połączenie, które korzysta z 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 korzystania z kanału transportowego HTTP. Jeśli połączenie zostanie pomyślnie utworzone, właściwość XMSC_WPM_CONNECTION_PROTOCOL połączenia zostanie ustawiona na wartość XMSC_WPM_CP_HTTP. Informacje na temat sposobu konfigurowania łańcuchów transportowych zawiera sekcja Konfigurowanie łańcuchów transportowych w dokumentacji produktu WebSphere Application Server .

Podczas nawiązywania połączenia z serwerem startowym aplikacja ma podobny wybór protokołów komunikacyjnych. Właściwość XMSC_WPM_PROVIDER_ENDPOINTS fabryki połączeń jest sekwencją jednego lub większej liczby adresów punktów końcowych serwerów startowych. Program startowy łańcucha transportowego każdego adresu punktu końcowego może być albo XMSC_WPM_BOOTSTRAP_TCP, dla połączenia TCP/IP z serwerem startowym lub XMSC_WPM_BOOTSTRAP_HTTP, dla połączenia, które używa protokołu HTTP.

Sesje

Sesja jest jednowątkowym kontekstem do wysyłania i odbierania komunikatów.

Aplikacja może korzystać z sesji w celu tworzenia komunikatów, producentów komunikatów, konsumentów komunikatów, przeglądarek kolejek i miejsc docelowych tymczasowych. 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) zasubskrybują ten sam temat, każdy otrzymuje kopię dowolnego komunikatu opublikowanego w tym temacie.

W przeciwieństwie do obiektu połączenia, obiekt Session nie może być jednocześnie używany w różnych wątkach. Tylko metoda Close Session obiektu Session może być wywoływana z wątku innego niż ten, który jest używany przez obiekt Session w danym momencie. Metoda Close Session kończy sesję i zwalnia wszystkie zasoby systemowe przydzielone do sesji.

Jeśli aplikacja musi przetwarzać komunikaty współbieżnie w więcej niż jednym wątku, aplikacja musi utworzyć sesję dla każdego wątku, a następnie użyć tej sesji dla każdej operacji wysyłania lub odbierania w obrębie tego wątku.

Sesje transakcyjne

Aplikacje produktu XMS mogą uruchamiać transakcje lokalne. *Transakcja lokalna* to transakcja, która obejmuje zmiany tylko w przypadku zasobów menedżera kolejek lub magistrali integracji usług, z którymi połączona jest aplikacja.

Informacje zawarte w tym temacie są istotne tylko wtedy, gdy aplikacja łączy się z menedżerem kolejek produktu IBM MQ lub z magistralą integracji usług produktu WebSphere Application Server . Informacje te nie mają znaczenia dla połączenia w czasie rzeczywistym z brokerem.

Aby uruchomić transakcje lokalne, aplikacja musi najpierw utworzyć sesję transakcyjne, wywołując metodę Create Session obiektu połączenia, określając jako parametr, że sesja jest transakcjami transakcjami. Następnie wszystkie komunikaty wysłane i odebrane w ramach sesji są pogrupowane w sekwencję transakcji. Transakcja kończy się, gdy aplikacja zatwierdza lub wycofuje komunikaty, które zostały wysłane i odebrane od momentu rozpoczęcia transakcji.

Aby zatwierdzić transakcję, aplikacja wywołuje metodę zatwierdzania obiektu sesji. Jeśli transakcja zostanie zatwierdzona, wszystkie komunikaty wysłane w ramach transakcji stają się dostępne do dostarczenia do innych aplikacji, a wszystkie komunikaty odebrane w ramach transakcji zostaną potwierdzone, tak 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 odrzucane przez serwer przesyłania komunikatów, a wszystkie komunikaty odebrane w ramach transakcji stają się dostępne do dostarczenia ponownie. W domenie typu punkt z punktem komunikaty, które zostały odebrane, są ponownie umieszczane w ich kolejkach i ponownie stają się widoczne dla innych aplikacji.

Nowa transakcja jest uruchamiana automatycznie, gdy aplikacja tworzy sesję transakcyjna lub wywołuje metodę zatwierdzania lub wycofania zmian. Oznacza to, że sesja transakcyjna zawsze ma aktywną transakcję.

Gdy aplikacja zamknie sesję transakcyjne, następuje niejawne wycofanie zmian. Jeśli aplikacja zamknie połączenie, dla wszystkich sesji transakcyjnych połączenia zostanie wykonane niejawne wycofanie zmian.

Transakcja jest w całości zawarta w ramach sesji transakcyjnej. Transakcja nie może obejmować sesji. Oznacza to, że aplikacja nie może wysyłać i odbierać komunikatów w dwóch lub większej liczby sesji transakcyjnych, a następnie zatwierdzać lub wycofywanie wszystkich tych działań jako jednej transakcji.

Pojęcia pokrewne

Potwierdzenie komunikatu

Każda sesja, która nie jest transakcowana, ma tryb potwierdzenia, który określa sposób, w jaki odbierane są komunikaty odbierane przez aplikację. Dostępne są trzy tryby potwierdzenia, a wybór trybu potwierdzania wpływa na konstrukcję aplikacji.

Dostarczanie komunikatów
Produkt XMS obsługuje trwałe i nietrwałe tryby dostarczania komunikatów, a także asynchroniczne i synchroniczne dostarczanie komunikatów.

Potwierdzenie komunikatu

Każda sesja, która nie jest transakcowana, ma tryb potwierdzenia, który określa sposób, w jaki odbierane są komunikaty odbierane przez aplikację. Dostępne są trzy tryby potwierdzenia, a wybór trybu potwierdzania wpływa na konstrukcję aplikacji.

Informacje zawarte w tym temacie są istotne tylko wtedy, gdy aplikacja łączy się z menedżerem kolejek produktu IBM MQ lub z magistralą integracji usług produktu WebSphere Application Server . Informacje te nie mają znaczenia dla połączenia w czasie rzeczywistym z brokerem.

Produkt XMS korzysta z tego samego mechanizmu do potwierdzania odbioru komunikatów, których używa JMS.

Jeśli sesja nie jest transakowana, sposób potwierdzania komunikatów odbieranych przez aplikację jest określany przez tryb potwierdzania sesji. Trzy tryby potwierdzenia są opisane w następujących akapitach:

XMSC_AUTO_ACKNOWLEDGE
Sesja automatycznie potwierdzi każdy komunikat otrzymany przez aplikację.

Jeśli komunikaty są dostarczane synchronicznie do aplikacji, sesja potwierdza otrzymanie komunikatu za każdym razem, gdy wywołanie Odbiór zakończy się pomyślnie.

Jeśli aplikacja pomyślnie odbierze komunikat, ale niepowodzenie uniemożliwia potwierdzenie wystąpienia, komunikat staje się dostępny do ponownego dostarczenia. Dlatego aplikacja musi mieć możliwość obsługi komunikatu, który został ponownie dostarczony.

XMSC_DUPS_OK_ACKNOWLEDGE
Sesja potwierdza komunikaty odebrane przez aplikację w czasie, gdy jest ona wybierana.

Użycie tego trybu potwierdzenia zmniejsza ilość pracy, jaką musi wykonać sesja, ale błąd, który uniemożliwia potwierdzenie komunikatu, może spowodować ponowne udostępnienie więcej niż jednego komunikatu do dostarczenia. Dlatego aplikacja musi mieć możliwość obsługi komunikatów, które są ponownie dostarczane.

POTWIERDZANIE Xmsc_client_acknowledge
Aplikacja potwierdza otrzymywane przez niego komunikaty, wywołując metodę Acknowledge klasy Message.

Aplikacja może potwierdzić odbiór każdego komunikatu indywidualnie lub może otrzymać partię komunikatów i wywołać metodę Acknowledge tylko dla ostatniego komunikatu, który otrzymuje. Gdy metoda Acknowledge jest nazywana wszystkimi wiadomościami otrzymanymi od czasu ostatniego wywołania metody, są one potwierdzane.

W połączeniu z dowolnym z tych trybów potwierdzania aplikacja może zatrzymać i ponownie uruchomić dostarczanie komunikatów w sesji, wywołując metodę Recover klasy Session. Wiadomości, których przyjęcie zostało wcześniej niepotwierdzone, są ponownie dostarczane. Mogą one jednak nie być dostarczane w tej samej kolejności, w jakiej zostały dostarczone wcześniej. W międzyczasie mogły zostać wysłane komunikaty o wyższym priorytecie, a niektóre z oryginalnych komunikatów mogły utraci ważność. W domenie typu punkt z punktem niektóre z oryginalnych komunikatów mogły zostać skonsumowane przez inną aplikację.

Aplikacja może określić, czy komunikat jest ponownie dostarczany, sprawdzając zawartość pola nagłówka JMSRedelivered komunikatu. Aplikacja wykonuje tę aplikację, wywołując metodę Get JMSRedelivered klasy Message.

Pojęcia pokrewne
Sesje transakcyjne
Aplikacje produktu XMS mogą uruchamiać transakcje lokalne. *Transakcja lokalna* to transakcja, która obejmuje zmiany tylko w przypadku zasobów menedżera kolejek lub magistrali integracji usług, z którymi połączona jest aplikacja.

Dostarczanie komunikatów
Produkt XMS obsługuje trwałe i nietrwałe tryby dostarczania komunikatów, a także asynchroniczne i synchroniczne dostarczanie komunikatów.

Dostarczanie komunikatów

Produkt XMS obsługuje trwałe i nietrwałe tryby dostarczania komunikatów, a także asynchroniczne i synchroniczne dostarczanie komunikatów.

Tryb dostarczania wiadomości

Produkt XMS obsługuje dwa tryby dostarczania komunikatów:

Trwałe
Komunikaty trwałe są dostarczane jeden raz. Serwer przesyłania komunikatów podejmuje specjalne środki ostrożności, takie jak rejestrowanie komunikatów, w celu zapewnienia, że trwałe komunikaty nie zostaną utracone podczas przesyłania, nawet w przypadku niepowodzenia.

Nietrwałe
Komunikaty nietrwałe 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 dostawy jest kompromisem między niezawodnością a wydajnością. Komunikaty nietrwałe są zwykle transportowane szybciej niż komunikaty trwałe.

Dostarczanie komunikatów asynchronicznych

Produkt XMS używa jednego wątku do obsługi wszystkich asynchronicznych dostaw komunikatów dla sesji. Oznacza to, że jednorazowo 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, to wszystkie inne konsumenty komunikatów, które oczekują na ten sam komunikat, muszą nadal czekać. Inne komunikaty, które oczekują na dostarczenie do sesji, muszą również czekać.

Jeśli aplikacja wymaga współbieżnego dostarczania komunikatów, utwórz więcej niż jedną sesję, tak aby produkt XMS używał więcej niż jednego wątku do obsługi asynchronicznego dostarczania komunikatów. W ten sposób współbieżnie może działać więcej niż jedna funkcja nasłuchiwania komunikatów lub metoda `onMessage()`.

Sesja nie jest asynchroniczna, przypisując obiekt nasłuchiwania komunikatów do konsumenta. Sesja staje się asynchroniczna tylko wtedy, gdy wywoływana jest metoda `Connection.Start`. Wszystkie wywołania synchroniczne są dozwolone, dopóki metoda `Connection.Start` nie zostanie wywołana. Dostarczanie komunikatów do konsumentów rozpoczyna się po wywołaniu `Connection.Start`.

Jeśli wywołania synchroniczne, takie jak tworzenie konsumenta lub producenta, muszą zostać wykonane 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 jest wątkiem, który dostarcza komunikaty do funkcji zwrotnej. Ten wątek może wywołać w funkcji zwrotnej komunikaty wywołanie sesji (z wyjątkiem wywołania Zamknij).

Uwaga: W trybie niezarządzanym wywołanie MQDISC w ramach funkcji połączenia telefocyjnego nie jest obsługiwane przez klienta IBM MQ .NET. Dlatego aplikacja kliencka nie może tworzyć ani zamykać sesji w ramach wywołania zwrotnego MessageListener w asynchronicznym trybie odbioru. Utwórz i rozdysponuj sesję poza metodą `MessageListener`.

Synchroniczne dostarczanie komunikatów

Komunikaty są dostarczane synchronicznie do aplikacji, jeśli w aplikacji używane są metody odbierania obiektów MessageConsumer.

Za pomocą metod odbierania aplikacja może czekać na określony czas dla komunikatu lub może czekać na czas nieokreślony. Alternatywnie, jeśli aplikacja nie chce czekać na komunikat, może skorzystać z metody receive bez oczekiwania.

Pojęcia pokrewne

Sesje transakcyjne
Aplikacje produktu XMS mogą uruchamiać transakcje lokalne. *Transakcja lokalna* to transakcja, która obejmuje zmiany tylko w przypadku zasobów menedżera kolejek lub magistrali integracji usług, z którymi połączona jest aplikacja.

Potwierdzenie komunikatu
Każda sesja, która nie jest transakcowana, ma tryb potwierdzenia, który określa sposób, w jaki odbierane są komunikaty odbierane przez aplikację. Dostępne są trzy tryby potwierdzenia, a wybór trybu potwierdzania wpływa na konstrukcję aplikacji.

Miejsca docelowe

Aplikacja XMS używa obiektu docelowego do określenia miejsca docelowego wysyłanych komunikatów oraz źródła komunikatów, które są odbierane.

Aplikacja XMS może albo utworzyć obiekt docelowy w czasie wykonywania, albo uzyskać predefiniowane miejsce docelowe z repozytorium administrowanych obiektów.

Podobnie jak w przypadku elementu ConnectionFactory, najbardziej elastycznym sposobem na określenie miejsca docelowego przez aplikację XMS jest zdefiniowanie go jako obiektu administrowanego. Korzystając z tego podejścia, aplikacje napisane w językach C, C + + i .NET oraz Javamogą współużytkować definicje miejsca docelowego. Właściwości administrowanych obiektów docelowych można zmieniać bez zmieniania dowolnego kodu.

W przypadku aplikacji .NET miejsce docelowe jest tworzone przy użyciu metody CreateTopic lub CreateQueue . Te dwie metody są dostępne zarówno w obiektach ISession, jak i XMSFactoryFactory w interfejsie API produktu .NET . Więcej informacji na ten temat zawierają sekcje “Miejsca docelowe w produkcie .NET” na stronie 661 i ../com.ibm.mq.ref.dev.doc/sapidest.dita#sapidest.

Identyfikatory ujednolicona zasobu tematu

Identyfikator URI (Uniform Resource Identifier) tematu określa nazwę tematu. Można również określić dla niego jedną lub więcej 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, można znaleźć w sekcji Właściwości miejsca docelowego.

Podczas określania identyfikatora URI tematu w celu użycia w subskrypcji można używać znaków wieloznacznych. Składnia dla tych znaków wieloznacznych zależy od typu połączenia i wersji brokera. Dostępne są następujące opcje:

- Menedżer kolejek produktu IBM WebSphere MQ 7.0 z użyciem znaków wieloznacznych na poziomie znaku
- Menedżer kolejek produktu IBM WebSphere MQ 7.0 z użyciem znaków wieloznacznych na poziomie tematu
- WebSphere Application Server magistrala integracji usług

Menedżer kolejek produktu IBM WebSphere MQ 7.0 z użyciem znaków wieloznacznych na poziomie znaku

W produkcie IBM WebSphere MQ 7.0 menedżer kolejek z użyciem znaków wieloznacznych na poziomie znaku używa następujących znaków wieloznacznych:

* dla 0 lub więcej znaków
? dla 1 znaku
% dla znaku zmiany znaczenia

W sekcji Tabela 81 na stronie 647 przedstawiono przykłady użycia tego schematu znaków wieloznacznych.

Tabela 81. Przykładowe identyfikatory URI korzystające ze znaków wieloznacznych na poziomie znaku dla menedżera kolejek produktu IBM WebSphere MQ 7.0

Jednolity identyfikator zasobu	**Zgodne**	**Przykłady**
"topic://Sport *Wyniki"	Wszystkie tematy zaczynając od "Sport" i kończąc w "Wyniki"	"topic://SportsResults" i "topic://Sport/Hockey/National/Div3/Results"
" topic://Sport?Wyniki "	Wszystkie tematy zaczynając od "Sport", po którym następuje jeden znak, po którym następuje "Results"	"topic://SportsResults" i "topic://SportXResults"
"topic://Sport/ * ball*/Div? / Results/*/???"	Tematy	"topic://Sport/Football/Div1/Results/2002/Nov" oraz "topic://Sport/Netball/National/Div3/Results/02/Jan"

Menedżer kolejek produktu IBM WebSphere MQ 7.0 z formatem dzikiego karty na poziomie tematu

Menedżer kolejek produktu IBM WebSphere MQ 7.0 z użyciem znaków wieloznacznych na poziomie tematu używa następujących znaków wieloznacznych:

aby dopasować wiele poziomów
+ dopasowanie do jednego poziomu

W programie Tabela 82 na stronie 647 przedstawiono przykłady użycia tego schematu znaków wieloznacznych.

Tabela 82. Przykładowe identyfikatory URI z użyciem schematu zastępczego na poziomie tematu dla menedżera kolejek produktu IBM WebSphere MQ 7.0

Jednolity identyfikator zasobu	**Zgodne**	**Przykłady**
"topic://Sport/ + /Wyniki"	Wszystkie tematy z pojedynczą hierarchiczną nazwą poziomu między Sport a Results	"topic://Sport/Football/Results" oraz "topic://Sport/Ju-Jitsu/Results"
"topic://Sport/#/Wyniki"	Wszystkie tematy zaczynając od "Sport/" i kończąc w "/Results"	"topic://Sport/Football/Results" oraz "topic://Sport/Hockey/National/Div3/Results"
" topic://Sport/Football/#"	Wszystkie tematy zaczynając od "Sport/Football/"	"topic://Sport/Football/Results" oraz "topic://Sport/Football/TeamNews/Signings/Managerial"

WebSphere Application Server magistrala integracji usług

Magistrala integracji usług produktu WebSphere Application Server korzysta z następujących znaków wieloznacznych:

*, aby dopasować dowolne znaki na jednym poziomie w hierarchii
// w celu dopasowania do 0 lub więcej poziomów
//. w celu dopasowania do wartości 0 lub większej liczby poziomów (na końcu wyrażenia tematu)

W programie Tabela 83 na stronie 648 przedstawiono przykłady użycia tego schematu znaków wieloznacznych.

Tabela 83. Przykładowe identyfikatory URI przy użyciu schematu zastępczego dla magistrali integracji usług produktu WebSphere Application Server

Jednolity identyfikator zasobu	**Zgodne**	**Przykłady**
"topic://Sport/ * ball/Wyniki"	Wszystkie tematy z pojedynczą hierarchiczną nazwą poziomu kończącą się na "ball" pomiędzy Sport a Results	"topic://Sport/Football/Results" oraz "topic://Sport/Netball/Results"
"topic://Sport//Wyniki"	Wszystkie tematy zaczynając od "Sport/" i kończąc w "/Results"	"topic://Sport/Football/Results" oraz "topic://Sport/Hockey/National/Div3/Results"
"topic://Sport/Football//."	Wszystkie tematy zaczynając od "Sport/Football/"	"topic://Sport/Football/Results" oraz "topic://Sport/Football/TeamNews/Signings/Managerial"
"topic://Sport/ * ball//Results//."	Tematy	"topic://Sport/Football/Results" oraz "topic://Sport/Netball/National/Div3/Results/2002/November"

Pojęcia pokrewne

Identyfikatory URI jednorodnych zasobów
Identyfikator URI kolejki określa nazwę kolejki. Można również określić jedną lub więcej właściwości kolejki.

Tymczasowe miejsca docelowe
Aplikacje produktu XMS mogą tworzyć tymczasowe miejsca docelowe i korzystać z nich.

Identyfikatory URI jednorodnych zasobów

Identyfikator URI kolejki określa nazwę kolejki. Można również określić jedną lub więcej właściwości kolejki.

Identyfikator URI dla kolejki rozpoczyna się od sekwencji `queue://`, po której następuje nazwa kolejki. Może ona również zawierać listę par nazwa-wartość, które ustawiają pozostałe właściwości kolejki.

W przypadku kolejek produktu IBM MQ (ale nie dla kolejek domyślnego dostawcy przesyłania komunikatów produktu WebSphere Application Server) menedżer kolejek, w którym rezyduje kolejka, może zostać określony przed kolejką, a nazwa menedżera kolejek zostanie oddzielona/oddzielająca od nazwy kolejki.

Jeśli określony jest menedżer kolejek, musi to być ten, do którego program XMS jest bezpośrednio połączony dla połączenia przy użyciu tej kolejki lub musi być dostępny z tej kolejki. Zdalne menedżery kolejek są obsługiwane tylko w przypadku pobierania komunikatów z kolejek, a nie w celu umieszczania komunikatów w kolejkach. Szczegółowe informacje na ten temat można znaleźć w dokumentacji menedżera kolejek produktu IBM MQ .

Jeśli nie został określony żaden menedżer kolejek, to dodatkowy/separator jest opcjonalny, a jego obecność lub brak nie ma znaczenia dla definicji kolejki.

Następujące definicje kolejek są równoważne dla kolejki IBM MQ o nazwie QB w menedżerze kolejek o nazwie QM_A, z którą XMS jest bezpośrednio połączone:

```
queue://QB
queue:///QB
queue://QM_A/QB
```

Pojęcia pokrewne

Identyfikatory ujednolicona zasobu tematu
Identyfikator URI (Uniform Resource Identifier) tematu określa nazwę tematu. Można również określić dla niego jedną lub więcej właściwości.

Tymczasowe miejsca docelowe
Aplikacje produktu XMS mogą tworzyć tymczasowe miejsca docelowe i korzystać z nich.

Tymczasowe miejsca docelowe

Aplikacje produktu XMS mogą tworzyć tymczasowe miejsca docelowe i korzystać z nich.

Aplikacja zwykle używa tymczasowego miejsca docelowego do odbierania odpowiedzi na komunikaty żądania. Aby określić miejsce docelowe, w którym ma zostać wysłana odpowiedź na komunikat żądania, aplikacja wywołuje metodę Set JMSReplyTo obiektu Message reprezentującego komunikat żądania. Miejsce docelowe określone w wywołaniu może być miejscem docelowym.

Chociaż sesja jest używana do tworzenia tymczasowego miejsca docelowego, zasięg tymczasowego miejsca docelowego jest w rzeczywistości połączeniem, który został użyty do utworzenia sesji. Każdy z sesji połączenia może tworzyć producentów komunikatów i konsumentów komunikatów dla tymczasowego miejsca docelowego. Tymczasowe miejsce docelowe pozostaje do czasu jawnego usunięcia lub nastąpi zakończenie połączenia, w zależności od tego, co nastąpi wcześniej.

Gdy aplikacja tworzy kolejkę tymczasową, tworzona jest kolejka na serwerze przesyłania komunikatów, z którym połączona jest aplikacja. Jeśli aplikacja jest połączona z menedżerem kolejek, tworzona jest kolejka dynamiczna z kolejki modelowej, której nazwa jest określona za pomocą właściwości 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, w magistrali tworzona jest kolejka tymczasowa, a przedrostek używany do tworzenia nazwy kolejki tymczasowej jest określony 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 ujednolicona zasobu tematu
Identyfikator URI (Uniform Resource Identifier) tematu określa nazwę tematu. Można również określić dla niego jedną lub więcej właściwości.

Identyfikatory URI jednorodnych zasobów
Identyfikator URI kolejki określa nazwę kolejki. Można również określić 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 należy określić poprawne miejsce docelowe podczas wysyłania komunikatu.

Producenci komunikatów z powiązanym miejscem docelowym

W tym scenariuszu producent komunikatów jest tworzony przy użyciu poprawnego miejsca docelowego. Podczas operacji wysyłania nie jest wymagane określenie miejsca docelowego.

Producenci komunikatów bez powiązanego miejsca docelowego

W programie XMS .NETproducent komunikatów może zostać utworzony z pustym miejscem docelowym.

Aby utworzyć producenta komunikatów bez powiązanego miejsca docelowego w przypadku korzystania z interfejsu API .NET , wartość NULL musi zostać przekazana jako parametr do metody `CreateProducer()` obiektu ISession (na przykład `session.CreateProducer(null)`). Jeśli komunikat jest wysyłany, należy jednak określić poprawne miejsce docelowe.

Konsumenci komunikatów

Konsumenty komunikatów można sklasyfikować jako trwałe i nietrwałe subskrybenty oraz odbiorców komunikatów synchronicznych i asynchronicznych.

Trwali subskrybenci

Trwały subskrybent to konsument komunikatów, który odbiera wszystkie komunikaty opublikowane w temacie, w tym komunikaty opublikowane w czasie, gdy subskrybent jest nieaktywny.

Informacje zawarte w tym temacie są istotne tylko wtedy, gdy aplikacja łączy się z menedżerem kolejek produktu IBM MQ lub z magistralą integracji usług produktu WebSphere Application Server . Informacje te nie mają znaczenia dla połączenia w czasie rzeczywistym z brokerem.

Aby utworzyć trwały subskrybent dla tematu, aplikacja wywołuje metodę Create Durable Subskrybenta dla obiektu Session, określając jako parametry nazwę identyfikującą trwałą subskrypcję i obiekt docelowy reprezentujący dany temat. Aplikacja może utworzyć trwały subskrybent z selektorem komunikatów lub bez niego, a także może określić, czy trwały subskrybent ma odbierać komunikaty publikowane przez własne połączenie.

Sesja użyta do utworzenia trwałego subskrybenta musi mieć powiązany identyfikator klienta. Identyfikator klienta jest taki sam jak identyfikator powiązany z połączeniem, który jest używany do utworzenia sesji. Jest on określony w sposób opisany w sekcji “Obiekty ConnectionFactories i obiekty Connection” na stronie 641.

Nazwa, która identyfikuje trwałą subskrypcję, musi być unikalna w obrębie identyfikatora klienta, dlatego identyfikator klienta jest częścią pełnego, unikalnego identyfikatora trwałej subskrypcji. Serwer przesyłania komunikatów przechowuje rekord trwałej subskrypcji i zapewnia, że wszystkie komunikaty publikowane w tym temacie są zachowywane aż do momentu ich potwierdzenia przez trwałą subskrybent lub utracą ważność.

Serwer przesyłania komunikatów nadal utrzymuje rekord trwałej subskrypcji, nawet po zamknięciu trwałego subskrybenta. Aby ponownie wykorzystać trwałą subskrypcję, która została wcześniej utworzona, aplikacja musi utworzyć trwały subskrybent, określając tę samą nazwę subskrypcji i przy użyciu sesji z tym samym identyfikatorem klienta, co powiązane z trwałą subskrypcją. Tylko jedna sesja w danym momencie może mieć trwały subskrybent dla konkretnej trwałej subskrypcji.

Zasięgiem trwałej subskrypcji jest serwer przesyłania komunikatów, który jest rejestrowany w ramach subskrypcji. Jeśli dwa aplikacje połączone z różnymi serwerami przesyłania komunikatów tworzą trwały subskrybent przy użyciu tej samej nazwy subskrypcji i identyfikatora klienta, tworzone są dwie całkowicie niezależne subskrypcje trwałe.

Aby usunąć trwałą subskrypcję, aplikacja wywołuje metodę Unsubscribe obiektu Session, określając jako parametr nazwę identyfikującą trwałą subskrypcję. Identyfikator klienta powiązany z sesją musi być taki sam, jak identyfikator powiązany z trwałą subskrypcją. Serwer przesyłania komunikatów usuwa rekord trwałej subskrypcji, która jest konserwowana, i nie wysyła kolejnych komunikatów do trwałego subskrybenta.

Aby zmienić istniejącą subskrypcję, aplikacja może utworzyć trwały subskrybent przy użyciu tej samej nazwy subskrypcji i identyfikatora klienta, ale podając inny temat lub selektor komunikatów (lub oba te elementy). Zmiana trwałej subskrypcji jest równoznaczna z usunięciem subskrypcji i utworzeniem nowej subskrypcji.

W przypadku aplikacji, która łączy się z menedżerem kolejek produktu IBM WebSphere MQ 7.0 lub nowszego, produkt XMS zarządza kolejkami subskrybentów. Dlatego aplikacja nie jest wymagana do określania kolejki subskrybenta. Produkt XMS zignoruje kolejkę subskrybenta, jeśli zostanie określona.

Należy pamiętać, że nie można zmienić kolejki subskrybenta dla trwałej subskrypcji. Jedynym sposobem, aby zmienić kolejkę subskrybenta, jest usunięcie subskrypcji i utworzenie nowej.

W przypadku aplikacji, która łączy się z magistralą integracji usług, każdy trwały subskrybent musi mieć wyznaczoną stałą subskrypcję trwałą. Aby określić katalog główny trwałej subskrypcji dla wszystkich stałych subskrybentów, które korzystają z tego samego połączenia, należy ustawić właściwość XMSC_WPM_DUR_SUB_HOME obiektu ConnectionFactory , który jest używany do tworzenia połączenia. Aby określić katalog główny trwałej subskrypcji dla danego tematu, należy ustawić właściwość XMSC_WPM_DUR_SUB_HOME obiektu docelowego reprezentującego dany temat. W przypadku połączenia przed utworzeniem trwałego subskrybenta, który korzysta z połączenia, musi zostać określony dom subskrypcji trwałej. Każda wartość określona dla miejsca docelowego przesłania wartość określoną dla połączenia.

Nietrwałe subskrybenty

Nietrwały subskrybent to konsument komunikatów, który odbiera tylko komunikaty, które są publikowane w czasie, gdy subskrybent jest aktywny. Komunikaty dostarczone w czasie, gdy subskrybent jest nieaktywny, są tracone.

Informacje zawarte w tym temacie są istotne tylko wtedy, gdy przesyłanie komunikatów w trybie publikowania/subskrypcji jest używane przez menedżer kolejek produktu IBM WebSphere MQ 6.0 .

Jeśli obiekty konsumenta nie zostaną usunięte przed lub w trakcie zamykania połączenia, komunikaty mogą pozostać w kolejkach brokera dla subskrybentów, które nie są już aktywne.

W takiej sytuacji kolejki te mogą zostać usunięte z tych komunikatów za pomocą programu narzędziowego do czyszczenia udostępnionego z klasami IBM WebSphere MQ classes for JMS dla usługi JMS. Szczegółowe informacje na temat korzystania z tego programu narzędziowego znajdują się w sekcji *IBM WebSphere MQ Using Java.* W przypadku dużej liczby komunikatów znajdujących się w tej kolejce konieczne może być również zwiększenie głębokości kolejki subskrybentów.

Synchroniczny i asynchroniczny konsument komunikatów

Synchroniczny konsument komunikatów odbiera komunikaty z kolejki synchronicznie, a asynchroniczny konsument komunikatów odbiera komunikat z kolejki asynchronicznie.

Synchroniczny konsument komunikatów

Konsument komunikatów synchronicznych odbiera jeden komunikat w danym momencie. Gdy używana jest metoda `Receive(wait interval)` , wywołanie oczekuje tylko określonego czasu (w milisekundach) dla komunikatu lub do momentu zamknięcia konsumenta komunikatu.

Jeśli używana jest metoda oczekiwania ReceiveNoWait (), konsument komunikatów synchronicznych odbiera komunikaty bez opóźnienia. Jeśli następny komunikat jest dostępny, zostanie odebrany natychmiast, w przeciwnym razie zostanie zwrócony wskaźnik do obiektu komunikatu o wartości NULL.

Konsumenty komunikatów asynchronicznych

Obiekt nasłuchiwania komunikatów zarejestrowany przez aplikację jest wywoływany za każdym razem, gdy w kolejce jest dostępny 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 i wcześniejsze wersje dla systemu Long Term Support, i IBM MQ 9.1.4 oraz wcześniejsze w przypadku systemu Continuous Delivery, i jego wcześniejszych wersji, jeśli kolejka lokalna lub kolejka docelowa dla kolejki aliasowej jest kolejką klastra, program 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.
- V 9.1.5 V 9.1.0.5 W przypadku używania wartości 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
Jeśli używane są narzędzia Application Server Facilities (ASF), ConnectionConsumer, a nie MessageConsumer, przetwarza komunikaty nieprzetwarzalne. Właściwości ConnectionConsumer są requiem komunikatów zgodnie z właściwościami QName BackoutThreshold i BackoutRequeuekolejki.

Jeśli aplikacja korzysta z opcji ConnectionConsumers, okoliczności, w których jest tworzona kopia zapasowa komunikatu, zależą od sesji, którą udostępnia serwer aplikacji:

- Gdy sesja jest nietransakowana, z AUTO_ACKNOWLEDGE lub DUPS_OK_ACKNOWLEDGE, komunikat jest wycofany tylko po wystąpieniu błędu systemowego lub nieoczekiwanie kończy działanie aplikacji.
- Gdy sesja nie jest transakowana za pomocą funkcji CLIENT_ACKNOWLEDGE, niepotwierdzone komunikaty mogą być wycofane przez serwer aplikacji wywołujący `Session.recover()`.

  Zazwyczaj implementacja klienta MessageListener lub serwer aplikacji wywołuje `Message.acknowledge()`. Program `Message.acknowledge()` potwierdza wszystkie komunikaty dostarczone do tej pory w sesji.
- Gdy sesja jest transmiana, komunikaty niepotwierdzone mogą być wycofane przez serwer aplikacji wywołujący `Session.rollback()`.

Przeglądarki kolejek

Aplikacja korzysta z przeglądarki kolejek w celu przeglądania komunikatów w kolejce bez usuwania ich.

Aby utworzyć przeglądarkę kolejek, aplikacja wywołuje metodę Create Queue Browser dla 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 kolejki aplikacja może wywołać metodę GetEnumerator obiektu IQueueBrowser w celu uzyskania listy komunikatów w kolejce. Metoda zwraca obiekt wyliczeniowy, który hermetykuje listę obiektów komunikatu. Kolejność obiektów komunikatu na liście jest taka sama, jak kolejność, w jakiej komunikaty będą pobierane z kolejki. Aplikacja może następnie użyć wyliczenia, aby przejrzeć każdy komunikat z kolei.

Program wyliczeniowy jest aktualizowany dynamicznie, ponieważ komunikaty są umieszczane w kolejce i usuwane z kolejki. Za każdym razem, gdy aplikacja wywołuje komendę IEnumerator.MoveNext() w celu przeglądania następnego komunikatu w kolejce, komunikat ten odzwierciedla bieżącą zawartość kolejki.

Aplikacja może wywołać metodę GetEnumerator więcej niż jeden raz dla danej przeglądarki kolejki. Każde wywołanie zwraca nowy obiekt wyliczeniowy. W związku z tym aplikacja może używać więcej niż jednego wyliczenia do przeglądania komunikatów w kolejce i obsługi wielu pozycji w kolejce.

Aplikacja może użyć przeglądarki kolejek w celu wyszukania odpowiedniego komunikatu do usunięcia z kolejki, a następnie użyć konsumenta komunikatów z selektorem komunikatów, aby usunąć ten 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 677.

Requestery

Aplikacja korzysta z requestera w celu wysłania komunikatu żądania, a następnie oczekiwania na odebranie odpowiedzi i odebrania 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 pomaga w opracowaniu tego stylu aplikacji.

W celu utworzenia requestera aplikacja wywołuje konstruktor żądania Create Requestor klasy Requestor, określając jako parametry obiekt Session i obiekt docelowy, który identyfikuje miejsca, w których mają być wysyłane komunikaty żądań. Sesja nie może być transakcjaowana ani nie ma trybu potwierdzania XMSC_CLIENT_ACKNOWLEDGE. Konstruktor automatycznie tworzy tymczasową kolejkę lub temat, w którym mają być wysyłane komunikaty odpowiedzi.

Po utworzeniu requestera aplikacja może wywołać metodę żądania obiektu Requestor w celu wysłania komunikatu żądania, a następnie oczekiwania na odpowiedź i odebranie odpowiedzi z aplikacji, która odbiera komunikat z żądaniem. Połączenie oczekuje na odebraną odpowiedź lub do momentu zakończenia sesji, w zależności od tego, co nastąpi wcześniej. Zamawiający wymaga tylko jednej odpowiedzi dla każdego komunikatu żądania.

Po zamknięciu przez aplikację requestera usuwana jest kolejka tymczasowa lub temat. Jednak powiązana sesja nie jest zamykana.

Usuwanie obiektu

Gdy aplikacja usunie utworzony obiekt XMS , produkt XMS zwalnia zasoby wewnętrzne, które zostały przydzielone do obiektu.

Gdy aplikacja tworzy obiekt XMS , produkt XMS przydziela pamięć i inne zasoby wewnętrzne do obiektu. Produkt XMS zachowuje te zasoby wewnętrzne do czasu, aż aplikacja jawnie usunie obiekt, wywołując metodę close lub delete obiektu, w którym punkt XMS zwalnia zasoby wewnętrzne. Jeśli aplikacja próbuje usunąć obiekt, który jest już usunięty, wywołanie jest ignorowane.

Gdy aplikacja usunie obiekt typu Połączenie lub Sesja, 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 niezależnie od obiektu. Obiekty te są wyświetlane w programie Tabela 84 na stronie 655.

Uwaga: Jeśli aplikacja zamknie połączenie z sesjami zależnymi, wszystkie obiekty zależne od tych sesji również zostaną usunięte. Obiekty zależne mogą być zależne tylko od obiektu połączenia lub sesji.

Tabela 84. Obiekty, które są automatycznie usuwane

Usunięty obiekt	**Metoda**	**Obiekty zależne, które są usuwane automatycznie**
Połączenie	Zamknij połączenie	Obiekty danych i sesji ConnectionMeta
Sesja	Zamknij sesję	Obiekty MessageConsumer, MessageProducer, QueueBrowseri Requestor

Zarządzanie transakcjami XA IBM MQ za pomocą XMS

Zarządzane transakcje IBM MQ XA mogą być używane przez produkt XMS.

Aby można było używać transakcji XA przy użyciu produktu XMS, należy utworzyć sesję transakcyjne. Gdy transakcja XA jest używana, sterowanie transakcjami odbywa się za pośrednictwem globalnych transakcji rozproszonych koordynatora transakcji (Distributed Transaction Coordinator-DTC) i nie jest to ani sesje produktu XMS . W przypadku korzystania z transakcji XA nie można wydać `Session.commit` lub `Session.rollback` w sesji XMS . 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, który został 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 dla produktu `Session.commit` lub `Session.rollback`.
- Obiekty producenta lub konsumenta, które są kiedyś używane w sesji transakcyjnej XA, są używane po stronie zasięgu transakcji XA.

Transakcje XA nie są obsługiwane w asynchronicznych konsumentach.

Uwaga:

1. Przed zatwierdzeniem transakcji XA możliwe jest wydanie zamknięcia na obiekcie `Producer`, `Consumer`, `Session`lub `Connection` . W takim przypadku komunikaty w transakcji są wycofywane. Podobnie, jeśli połączenie zostało zerwane przed zatwierdzeniem transakcji XA, wszystkie komunikaty w transakcji są wycofywane. W przypadku obiektu `Producer` wycofanie oznacza, że komunikaty nie są umieszczane w kolejce. W przypadku obiektu `Consumer` wycofanie oznacza, że komunikaty pozostają w kolejce.
2. Jeśli obiekt `Producer` wstawi komunikat `TimeToLive` (Czas życia) w `TransactionScope` , a `commit` zostanie wystawiony po upłynięciu czasu, komunikat może utracyć ważność przed wydaniem `commit` . W tym przypadku komunikat nie jest dostępny dla obiektów `Consumer` .

3. Obiekty `Session` nie są obsługiwane przez wątki. Użycie transakcji z obiektami `Session` , które są współużytkowane przez wątki, nie jest obsługiwane.

Typy podstawowe produktu XMS

Produkt XMS udostępnia odpowiedniki ośmiu typów podstawowych Java (byte, short, int, long, float, double, char i boolean). Pozwala to na wymianę komunikatów między XMS a JMS bez utraty lub uszkodzenia danych.

Tabela 85 na stronie 656 zawiera Java równoważny typ danych, wielkość oraz minimalną i maksymalną wartość każdego typu podstawowego XMS .

Tabela 85. 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	true
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)
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 (z dokładnością do 7 cyfr)	3.402823E+38 (z dokładnością do 7 cyfr)
System.Double	double (podwójna)	64 bity	-1.79769313486231E-308 (do 15 -cyfrowej precyzji)	1.79769313486231E+308 (do 15 -cyfrowej precyzji)

Niejawne przekształcenie wartości właściwości z jednego typu danych na inny.

Gdy aplikacja pobiera wartość właściwości, wartość może zostać przekształcona przez produkt XMS na inny typ danych. Wiele reguł decyduje o tym, które konwersje są obsługiwane i w jaki sposób program XMS wykonuje konwersje.

Właściwość obiektu ma nazwę i wartość. Wartość ma powiązany typ danych, w którym wartość właściwości jest określana także jako *typ właściwości*.

Aplikacja korzysta z metod klasy PropertyContext w celu pobrania i ustawienia właściwości obiektów. W celu uzyskania wartości właściwości aplikacja wywołuje metodę odpowiednią dla typu właściwości. Na przykład, aby uzyskać wartość właściwości całkowitej, aplikacja zwykle wywołuje metodę GetIntProperty.

Jeśli jednak aplikacja pobiera wartość właściwości, wartość może zostać przekształcona przez produkt XMS na inny typ danych. Na przykład, aby uzyskać wartość właściwości całkowitej, aplikacja może wywołać metodę GetStringProperty, która zwraca wartość właściwości jako łańcuch. Konwersje obsługiwane przez produkt XMS są wyświetlane w programie Tabela 86 na stronie 657.

Tabela 86. 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 określają następujące reguły ogólne:

- Wartości właściwości liczbowych mogą być przekształcane z jednego typu danych na inny, pod warunkiem, że podczas konwersji nie są tracone żadne dane. Na przykład wartość właściwości o typie danych System.Int32 może zostać przekształcona w wartość o typie danych System.Int64, ale nie może zostać przekształcona w wartość o typie danych System.Int16.
- Wartość właściwości dowolnego typu danych może zostać przekształcona w łańcuch.
- Wartość właściwości łańcuchowej może zostać przekształcona w dowolny inny typ danych pod warunkiem, że łańcuch zostanie poprawnie sformatowany w celu konwersji. Jeśli aplikacja podejmie próbę przekształcenia wartości właściwości łańcuchowej, która nie jest poprawnie sformatowana, program XMS może zwrócić błędy.
- Jeśli aplikacja podejmie próbę konwersji, która nie jest obsługiwana, program XMS może zwrócić błąd.

Następujące reguły mają zastosowanie, gdy wartość właściwości jest przekształcana z jednego typu danych na inny:

- Podczas konwersji wartości właściwości boolowskiej na łańcuch wartość true jest przekształcana w łańcuch "true", a wartość false jest przekształcana w łańcuch "false".
- Podczas konwersji wartości właściwości boolowskiej na liczbowy typ danych, w tym System.SByte, wartość true jest przekształcana na 1, a wartość false jest przekształcana na 0.
- Podczas przekształcania wartości właściwości łańcuchowej na wartość boolowskim łańcuch "true" (bez rozróżniania wielkości liter) lub "1" jest przekształcany na wartość true, a łańcuch "false" (bez rozróżniania wielkości liter) lub "0" jest przekształcany na wartość false. Nie można przekształcić wszystkich pozostałych łańcuchów.
- Podczas konwersji wartości właściwości łańcuchowej na wartość o typie danych System.Int32, System.Int64, System.SBytelub System.Int16łańcuch musi mieć następujący format:

  [*odstępy*] [*znak*]*cyfry*

  Komponenty łańcuchowe są definiowane w następujący sposób:

  wartości puste
  Opcjonalne wiodące puste znaki.

  sign
  Opcjonalny znak plus (+) lub znak minus (-).

  cyfry
  Ciągła sekwencja znaków cyfr (0-9). Musi istnieć co najmniej jeden znak cyfry.

Po sekwencji znaków cyfr łańcuch może zawierać inne znaki, które nie są znakami cyfr, ale konwersja zatrzymuje się, gdy tylko pierwszy z tych znaków zostanie osiągnięty. Przyjmuje się, że łańcuch reprezentuje dziesiętną liczbę całkowitą.

XMS może zwrócić błąd, jeśli łańcuch nie jest poprawnie sformatowany.

- Podczas przekształcania wartości właściwości łańcuchowej na wartość typu danych System.Double lub System.Floatłańcuch musi mieć następujący format:

  [*odstępy*] [*znak*] [*cyfry*] [*punkt*[*d_cyfry*]] [*znak e_char*[*znak*]*cyfra_cyfry*]

  Komponenty łańcuchowe są definiowane w następujący sposób:

  wartości puste
  (Opcjonalnie) Leading pustych znaków.

  sign
  (Opcjonalnie) Znak plus (+) lub znak minus (-).

  cyfry
  Ciągła sekwencja znaków cyfr (0-9). Co najmniej jedna cyfra musi być obecna w postaci *cyfr* lub *d_cyfr*.

  punkt
  (Opcjonalne) Punkt dziesiętny (.).

  d_cyfry
  Ciągła sekwencja znaków cyfr (0-9). Co najmniej jedna cyfra musi być obecna w postaci *cyfr* lub *d_cyfr*.

  znak
  Znak wykładnika, który ma wartość *E* lub *e*.

  znak
  (Opcjonalnie) Znak plus (+) lub znak minus (-) dla wykładnika.

  _cyfry
  Ciągła sekwencja znaków cyfr (0-9) dla wykładnika. Jeśli łańcuch zawiera znak wykładnika, musi być obecny co najmniej jeden znak cyfry.

  Po sekwencji znaków cyfr lub opcjonalnych znaków reprezentujących wykładnik, łańcuch może zawierać inne znaki, które nie są znakami cyfr, ale konwersja zatrzymuje się, gdy tylko pierwszy z tych znaków zostanie osiągnięty. Przyjmuje się, że łańcuch reprezentuje liczbę dziesiętną zmiennopozycyjną z wykładnikiem, który jest potęgową liczbą 10.

  XMS może zwrócić błąd, jeśli łańcuch nie jest poprawnie sformatowany.

- Podczas przekształcania wartości liczbowej właściwości w łańcuch, w tym wartość właściwości o typie danych System.SByte, wartość ta jest przekształcana w łańcuchową reprezentację wartości jako liczby dziesiętnej, a nie łańcucha zawierającego znak ASCII dla tej wartości. Na przykład liczba całkowita 65 jest przekształcana w łańcuch "65", a nie na łańcuch "A".
- Przekształcając wartość właściwości tablicy bajtów na łańcuch, każdy bajt jest przekształcany w 2 znaki szesnastkowe, które reprezentują bajt. Na przykład tablica bajtów {0xF1, 0x12, 0x00, 0xFF} jest konwertowana na łańcuch "F11200FF".

Konwersje z typu właściwości do innych typów danych są obsługiwane przez metody klas właściwości i PropertyContext .

Iteratory

Iterator hermetyzuje listę obiektów i kursor, który utrzymuje bieżącą pozycję na liście. Pojęcie iteratora (dostępnego w wersji IBM Message Service Client for C/C++) jest implementowane za pomocą interfejsu IEnumerator w produkcie IBM Message Service Client for .NET.

Gdy tworzony jest iterator, pozycja kursora jest przed pierwszym obiektem. Aplikacja korzysta z iteratora w celu pobrania każdego z nich z kolei.

Klasa Iterator IBM Message Service Client for C/C++ jest równoważna klasie Enumerator w produkcie Java. IBM Message Service Client for .NET jest podobny do Java i korzysta z interfejsu IEnumerator.

Aplikacja może używać numeru IEnumerator do wykonywania następujących zadań:

- Aby uzyskać właściwości komunikatu
- Aby uzyskać pary nazwa-wartość w treści komunikatu mapy
- Przeglądanie komunikatów w kolejce
- Aby uzyskać nazwy zdefiniowanych właściwości komunikatu JMS obsługiwanych przez połączenie,

Identyfikatory kodowanego zestawu znaków

W programie XMS .NETwszystkie łańcuchy są przekazywane za pomocą rodzimego łańcucha .NET . Ze względu na to, że jest to kodowanie stałe, nie są wymagane żadne dodatkowe informacje w celu jego interpretacji. Dlatego właściwość XMSC_CLIENT_CCSID nie jest wymagana dla aplikacji XMS .NET .

Kody błędów i wyjątków produktu XMS

Produkt XMS używa szeregu kodów błędów do wskazania niepowodzeń. Te kody błędów nie są jawnie wymienione w tej dokumentacji, ponieważ mogą się one różnić w zależności od wydania do wydania. Tylko kody wyjątków XMS (w formacie XMS_X_ ...) są udokumentowane, ponieważ pozostają one takie same w różnych wersjach produktu XMS.

Automatyczne ponowne połączenie klienta IBM MQ za pomocą XMS

Skonfiguruj klienta produktu XMS w taki sposób, aby ponownie nawiązał połączenie po awarii sieci, menedżera kolejek lub serwera podczas korzystania z klienta IBM WebSphere MQ 7.1 , a następnie jako dostawcy komunikatów.

Użyj właściwości `WMQ_CONNECTION_NAME_LIST` i `WMQ_CLIENT_RECONNECT_OPTIONS` klasy `MQConnectionFactory` , aby skonfigurować połączenie klienta w celu automatycznego ponownego połączenia. Automatyczne ponowne połączenie klienta ponownie łączy klienta po awarii połączenia lub jako opcja po zatrzymaniu menedżera kolejek. Projektowanie niektórych aplikacji klienckich sprawia, że nie nadają się one do automatycznego ponownego połączenia.

Po nawiązaniu połączenia automatycznie połączalne połączenia klienckie stają się ponownie połączalne.

Uwaga: Właściwości `Client Reconnect Options`(Opcje ponownego połączenia klienta), `Client Reconnect Timeout`(Limit czasu ponownego połączenia klienta) i `Connection Namelist` (Lista nazw połączeń) można również ustawić za pomocą tabeli definicji kanału klienta (Client Channel Definitions Table-CCDT) lub przez włączenie ponownego połączenia klienta przy użyciu pliku `mqclient.ini` .

Uwaga: Jeśli właściwości ponownego połączenia są ustawione w obiekcie `ConnectionFactory` i podobnie jak w tabeli definicji kanału klienta, reguła kolejności wykonywania jest następująca. Jeśli wartość domyślna właściwości listy nazw połączeń jest ustawiona w obiekcie `ConnectionFactory` , to środowisko CCDT ma pierwszeństwo. 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 produktu XMS .NET

Tematy zawarte w tej sekcji zawierają informacje pomocne podczas pisania aplikacji XMS .NET .

O tym zadaniu

Ta sekcja zawiera informacje specyficzne dla tworzenia aplikacji programu XMS .NET . Ogólne informacje na temat pisania aplikacji XMS zawiera sekcja “Pisanie aplikacji produktu XMS” na stronie 639.

Sekcja obejmuje następujące tematy:

- “Typy danych dla .NET” na stronie 660

- “Operacje zarządzane i niezarządzane w programie .NET” na stronie 661
- “Miejsca docelowe w produkcie .NET” na stronie 661
- “Właściwości w programie .NET” na stronie 662
- “Obsługa właściwości nieistniejących w produkcie .NET” na stronie 663
- “Obsługa błędów w produkcie .NET” na stronie 663
- “Korzystanie z programów nasłuchujących komunikatów i wyjątków w produkcie .NET” na stronie 663

Typy danych dla .NET

Program 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 dla XMS .NET różnią się od typów danych dla XMS C/C + +. Można użyć tego tematu w celu zidentyfikowania odpowiednich typów danych.

W poniższej tabeli przedstawiono odpowiadające im typy danych XMS .NET i XMS C/C + +, a w skrócie opisano je w skrócie.

Tabela 87. Typy danych dla systemów XMS .NET i XMS C/C++

Typ XMS .NET	**XMS Typ C/C++**	**Opis**
System.SByte	xmsSBYTE xmsINT8	Podpisana 8-bitowa wartość
System.Byte	xmsBYTE xmsUINT8	Wartość 8-bitowa bez znaku
System.Int16	xmsINT16 xmsSHORT	16-bitowa wartość ze znakiem
System.UInt16	xmsUINT16 xmsUSHORT	16-bitowa wartość bez znaku
System.Int32	xmsINT32 xmsINT	Podpisana 32-bitowa wartość
System.UInt32	xmsUINT32 xmsUINT	32-bitowa wartość bez znaku
System.Int64	xmsLONG xmsINT64	Podpisana 64-bitowa wartość
System.UInt64	xmsULONG xmsUINT64	64-bitowa wartość bez znaku
System.Char	xmsCHAR16	16-bitowy znak bez znaku (Unicode dla środowiska .NET)
System.Single	xmsFLOAT	Liczba 32-bitowych liczb zmiennopozycyjnych IEEE
System.Double	xmsDOUBLE	Liczba zmiennopozycyjna IEEE 64-bit
System.Boolean	xmsBOOL	Wartość Prawda/Fałsz

Tabela 87. Typy danych dla systemów XMS .NET i XMS C/C++ (kontynuacja)		
Typ XMS .NET	**XMS Typ C/C++**	**Opis**
Nie dotyczy	xmsCHAR	Podpisana lub niepodpisana wartość 8-bitowa (podpisana lub niepodpisana zależy od platformy)
System.Decimal	Nie dotyczy	96-bit podpisanych liczb całkowitych od 10^0 do 10^{28}
System.Object	Nie dotyczy	Podstawa wszystkich typów
System.String	Nie dotyczy	Typ łańcuchowy

Operacje zarządzane i niezarządzane w programie .NET

Kod zarządzany jest wykonywany wyłącznie w środowisku wykonawczym języka wspólnego produktu .NET i jest w pełni zależny od usług udostępnianych przez to środowisko wykonawcze. Aplikacja jest klasowana jako niezarządzana, jeśli dowolna część aplikacji działa lub wywołuje usługi poza środowiskiem wykonawczym wspólnego języka produktu .NET .

Niektóre zaawansowane funkcje nie mogą obecnie być obsługiwane w zarządzanym środowisku produktu .NET .

Jeśli aplikacja wymaga pewnych funkcji, które nie są obecnie obsługiwane w pełnym środowisku zarządzanym, można zmienić aplikację tak, aby korzystała z niezarządzanego środowiska bez konieczności wprowadzania istotnych zmian w aplikacji. Należy jednak pamiętać, że stos XMS korzysta z kodu niezarządzanego po dokonaniu wyboru.

Połączenia z menedżerem kolejek produktu IBM MQ

Połączenia zarządzane z programem WMQ_CM_CLIENT nie obsługują komunikacji innej niż TCP, a kompresja kanału. Połączenia te mogą jednak być obsługiwane przy użyciu połączenia niezarządzanego (WMQ_CM_CLIENT_UNMANAGED). Więcej informacji na ten temat zawiera sekcja Tworzenie aplikacji .NET.

Jeśli fabryka połączeń zostanie utworzona z obiektu administrowanego w środowisku niezarządzanym, należy ręcznie zmienić wartość dla 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 produktu WebSphere Application Server , które wymagają użycia protokołu SSL (w tym protokołu HTTPS), nie są obecnie obsługiwane jako kod zarządzany.

Miejsca docelowe w produkcie .NET

W programie .NETmiejsca docelowe są tworzone zgodnie z typem protokołu i mogą być używane tylko w przypadku typu protokołu, dla którego zostały utworzone.

Udostępniono dwie funkcje służące do tworzenia miejsc docelowych, jeden dla tematów i jeden dla kolejek:

- `IDestination CreateTopic(String topic);`
- `IDestination CreateQueue(String queue);`

Funkcje te są dostępne w następujących dwóch obiektach interfejsu API:

- ISesja
- XMSFactoryFactory

W obu przypadkach metody te mogą akceptować łańcuch w stylu URI, który może zawierać parametry, w następującym formacie:

```
"topic://some/topic/name?priority=5"
```

Alternatywnie metody te mogą akceptować tylko nazwę miejsca docelowego, to znaczy nazwę bez tematu: // lub kolejkę: // przedrostek i bez parametrów.

Oznacza to, że następujący łańcuch stylu URI:

```
CreateTopic("topic://some/topic/name");
```

spowodowałoby taki sam wynik, jak następująca nazwa docelowa:

```
CreateTopic("some/topic/name");
```

Podobnie jak w przypadku WebSphere Application Server magistrali integracji usług JMS, tematy mogą być również określane w postaci skróconych, która zawiera zarówno *topicname* , jak i *topicspace* , ale nie może zawierać parametrów:

```
CreateTopic("topicspace:topicname");
```

Właściwości w programie .NET

Aplikacja .NET korzysta z metod w interfejsie PropertyContext do pobierania i ustawiania właściwości obiektów.

Interfejs PropertyContext hermetyzuje metody, które dostają i ustawiają 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 zastępuje poprzednią wartość posiadanej właściwości.

Więcej informacji na temat właściwości produktu XMS zawiera sekcja Właściwości obiektów XMS.

W przypadku łatwości używania nazwy właściwości i wartości właściwości XMS w produkcie XMS są predefiniowane jako stałe publiczne w strukturze o nazwie XMSC. Nazwy tych stałych są w formacie XMSC.*stała*, na przykład XMSC.USERID (stała nazwa właściwości) i XMSC.DELIVERY_AS_APP (stała wartość).

Dodatkowo można uzyskać dostęp do stałych IBM MQ za pomocą IBM.XMS.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 istnieje aplikacja hybrydowa, która używa zarówno klas XMS .NET , jak i IBM MQ dla produktu .NET , a importuje to zarówno IBM.XMS i IBM.WMQ , a następnie należy w pełni kwalifikować przestrzeń nazw struktury MQC, aby upewnić się, że każde wystąpienie jest unikalne.

Niektóre zaawansowane funkcje nie są obecnie obsługiwane w zarządzanym środowisku .NET . Więcej informacji na ten temat zawiera sekcja “Operacje zarządzane i niezarządzane w programie .NET” na stronie 661 .

Obsługa właściwości nieistniejących w produkcie .NET

Obsługa nieistniejących właściwości w XMS .NET jest zasadniczo zgodna ze specyfikacją JMS, a także z implementacjami C i C + + XMS.

W usłudze JMS uzyskiwanie 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:

- Właściwość getStringi właściwość getObjectzwracają wartość NULL
- getBooleanWłaściwość zwraca wartość false, ponieważ wartość Boolean.valueOf(null) zwraca wartość false.
- getIntWłaściwość etc.throw java.lang.NumberFormatException , ponieważ metoda Integer.valueOf(null) zgłasza wyjątek

Jeśli właściwość nie istnieje w programie XMS .NET , występują następujące wyjątki:

- Właściwość GetStringi właściwość GetObject(i właściwość GetBytes) zwracają wartość NULL (co jest takie samo, jak właściwość Java).
- GetBooleanWłaściwość zgłasza wyjątek System.NullReferenceException
- Właściwość GetIntpowoduje zgłoszenie wyjątku System.NullReferenceException .

Ta implementacja różni się od Java, ale jest ona ogólnie spójna ze specyfikacją JMS oraz z interfejsami XMS C i C++. Podobnie jak w przypadku implementacji Java , program XMS .NET propaguje wszystkie wyjątki z wywołania System.Convert do programu wywołującego. Jednak w przeciwieństwie do produktu Java , produkt XMS jawnie zgłasza wyjątki NullReference, a nie tylko za pomocą rodzimego zachowania środowiska .NET przez przekazanie wartości NULL do procedur konwersji systemu. Jeśli aplikacja ustawia właściwość na łańcuch "abc" i wywołuje właściwość GetInt, wyjątek System.FormatException zgłaszany przez wartość Convert.ToInt32("abc") jest propagowany do programu wywołującego, który jest spójny z programem Java. Wyjątek MessageFormatjest zgłaszany tylko wtedy, gdy typy używane dla właściwości setProperty i getProperty są niezgodne. To zachowanie jest również spójne z produktem Java.

Obsługa błędów w produkcie .NET

Wyjątki XMS .NET wywodzi się z wyjątku System.Exception. Wywołania metod XMS mogą zgłaszać konkretne wyjątki XMS , takie jak MessageFormatException, general XMSExceptions lub systemowe wyjątki, takie jak NullReferenceException.

Aplikacje napisz, aby wychwytywać dowolne z tych błędów, albo w określonych blokach catch, albo w ogólnych blokach catch `System.Exception` , zgodnie z wymaganiami aplikacji.

Korzystanie z programów nasłuchujących komunikatów i wyjątków w produkcie .NET

Aplikacja .NET używa obiektu nasłuchiwania komunikatów w celu asynchronicznego odbierania komunikatów i asynchronicznego powiadamiania o problemie z połączeniem korzysta z obiektu nasłuchiwania wyjątków.

O tym zadaniu

Funkcjonalność programów nasłuchujących komunikatów i wyjątków jest taka sama dla produktu .NET , jak i dla języka C + +. Jednak niewielkie różnice w implementacji są niewielkie.

Procedura

- Aby skonfigurować program nasłuchujący komunikatów w celu asynchronicznego odbierania komunikatów, wykonaj następujące kroki:
  a) Zdefiniuj metodę, która jest zgodna z sygnaturą delegata programu nasłuchującego komunikatów.
  Definiowana metoda może być metodą statyczną lub instancją 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, używając czegoś podobnego do następującego przykładu:

```
MessageListener OnMsgMethod = new MessageListener(SomeMethodName)
```

  c) Zarejestruj delegata z jednym lub większą liczbę konsumentów, ustawiając go w właściwości MessageListener konsumenta:

```
consumer.MessageListener = OnMsgMethod;
```

  Aby usunąć delegata, należy przywrócić wartość NULL przez ustawienie parametru MessageListener :

```
consumer.MessageListener = null;
```

- Aby skonfigurować obiekt nasłuchiwania wyjątków, wykonaj następujące kroki.
  Obiekt nasłuchiwania wyjątków działa w taki sam sposób, jak program nasłuchujący komunikatów, ale ma inną definicję delegowania i jest przypisywany do połączenia, a nie przez konsumenta komunikatów. Jest to takie samo, jak w przypadku 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, używając czegoś podobnego do następującego przykładu:

```
ExceptionListener OnExMethod = new ExceptionListener(SomeMethodName)
```

  c) Zarejestruj delegata za pomocą połączenia, ustawiając jego właściwość ExceptionListener :

```
connection.ExceptionListener = OnExMethod ;
```

Istnieje możliwość usunięcia delegata przez zresetowanie obiektu ExceptionListener w celu:

```
null:  connection.ExceptionListener = null;
```

Praca z administrowanymi obiektami XMS .NET

Tematy w tej sekcji zawierają informacje na temat administrowanych obiektów. Aplikacje produktu XMS mogą pobierać definicje obiektów z repozytorium administrowanych obiektów centralnych i używać ich do tworzenia fabryk połączeń i miejsc docelowych.

O tym zadaniu

Ta sekcja zawiera informacje pomocne w tworzeniu obiektów administrowanych i zarządzaniu nimi, opisując typy administrowanych repozytorium obiektów obsługiwanych przez produkt XMS . W tej sekcji wyjaśniono również, w jaki sposób aplikacja XMS nawiąże połączenie z repozytorium obiektów administrowanych w celu pobrania wymaganych obiektów administrowanych.

Sekcja obejmuje następujące tematy:

Obsługiwane typy repozytorium obiektów administrowanych przez produkt XMS .NET

Obiekty administrowane przez system plików i LDAP mogą być używane do łączenia się z IBM MQ i WebSphere Application Server, podczas gdy nazwy COS mogą być używane tylko do łączenia się z serwerem WebSphere Application Server.

Katalogi obiektów systemu plików przyjmują postać serializowanych obiektów Java Naming Directory Interface (JNDI). Katalogi obiektów LDAP to katalogi, które zawierają obiekty JNDI . Katalogi obiektów systemu plików i LDAP można administrować za pomocą narzędzia JMSAdmin dostarczanego z produktem IBM WebSphere MQ 6.0lub IBM MQ Explorer, który jest dostarczany z produktem IBM WebSphere MQ 7.0 i nowszym. Zarówno system plików, jak i katalogi obiektów LDAP mogą być używane do administrowania połączeniami klientów poprzez scentralizowanie fabryk połączeń i miejsc docelowych produktu IBM WebSphere MQ . Administrator sieci może wdrożyć wiele aplikacji, które odwołują się do tego samego centralnego repozytorium, i które są automatycznie aktualizowane w celu odzwierciedlenia zmian ustawień połączeń wprowadzonych w centralnym repozytorium.

Katalog nazw COS zawiera fabryki połączeń i miejsca docelowe produktu WebSphere Application Server service integration bus i może być administrowany przy użyciu Konsoli administracyjnej serwera WebSphere Application Server . Aby aplikacja XMS mogła pobierać obiekty z katalogu nazw COS, musi zostać wdrożona usługa Web Service wyszukiwania produktu JNDI . Ta usługa Web Service nie jest dostępna na wszystkich serwerach WebSphere Application Server service integration technologies. Szczegółowe informacje można znaleźć w dokumentacji produktu.

Uwaga: Zrestartuj połączenia aplikacji, aby zmiany wprowadzone w katalogu obiektów zostały uwzględnione.

Odwzorowanie właściwości XMS .NET dla administrowanych obiektów

Aby umożliwić aplikacjom produktu XMS .NET korzystanie z fabryki połączeń IBM MQ JMS i WebSphere Application Server oraz definicji obiektów docelowych, właściwości pobrane z tych definicji muszą być odwzorowane na odpowiednie właściwości produktu XMS , które można ustawić w fabrykach połączeń i miejscach docelowych produktu XMS .

Aby utworzyć na przykład fabrykę połączeń produktu XMS z właściwościami pobraną z fabryki połączeń JMS produktu IBM MQ , właściwości muszą być odwzorowane między tymi dwoma właściwościami.

Wszystkie odwzorowania właściwości są wykonywane automatycznie.

Tabela 88 na stronie 666 demonstruje odwzorowania między niektórymi najczęściej spotykanych właściwości fabryk połączeń i miejsc docelowych. Właściwości wyświetlane w tej tabeli są tylko małym zestawem przykładów, a nie wszystkie wyświetlane właściwości są istotne dla wszystkich typów połączeń i serwerów.

Tabela 88. Przykłady odwzorowania nazw dla właściwości fabryki połączeń i miejsca docelowego

Nazwa właściwości IBM MQ JMS	**XMS Nazwa właściwości**	**WebSphere Application Server service integration bus Nazwa właściwości**
TRWAŁOŚĆ (PER)	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 wyświetlane w programie Tabela 89 na stronie 666 mają zastosowanie zarówno w przypadku usługi JMS, jak i dla produktu XMS .NET.

Tabela 89. Właściwości produktu XMS .NET

Właściwość	**Typ obiektu**				
	CF	**QCF**	**TCF**	**Kolejka**	**Temat**
APPLICATIONNAME	Y	Y	Y	N/A	N/A
ASYNCEXCEPTION	Y	Y	Y	N/A	N/A
CCDTURL	Y	Y	Y	N/A	N/A
CHANNEL	Y	Y	Y	N/A	N/A
CONNECTIONNAMELIST	Y	Y	Y	N/A	N/A
CLIENTRECONNECTOPTIONS	Y	Y	Y	N/A	N/A
CLIENTRECONNECTTIMEOUT	Y	Y	Y	N/A	N/A

Tabela 89. Właściwości produktu XMS .NET (kontynuacja)					
Właściwość	**Typ obiektu**				
	CF	**QCF**	**TCF**	**Kolejka**	**Temat**
CLIENTID	N/A	Y	N/A	N/A	N/A
COMPHDR "1" na stronie 667	Y	N/A	Y	N/A	N/A
COMPMSG "1" na stronie 667	Y	Y	Y	N/A	N/A
CONNOPT "1" na stronie 667	Y	Y	Y	N/A	N/A
CONNTAG "1" na stronie 667	Y	Y	Y	N/A	N/A
Opis "1" na stronie 667	N/A	Y	N/A	Y	Y
WAŻNOŚCI "1" na stronie 667	N/A	N/A	N/A	Y	Y
FAILIFQUIESCE	Y	Y	Y	Y	Y
nazwa_hosta	N/A	Y	N/A	N/A	N/A
LOCALADDRESS	N/A	Y	N/A	N/A	N/A
PERSISTENCE	N/A	N/A	N/A	Y	Y
PORT	N/A	Y	N/A	N/A	N/A
Priorytet"1" na stronie 667	N/A	N/A	N/A	Y	Y
PROVIDERVERSION "1" na stronie 667	N/A	Y	N/A	N/A	N/A
QMANAGER	Y	Y	Y	Y	N/A
kolejka"1" na stronie 667	N/A	N/A	N/A	Y	N/A
SHARECONVALLOWED	Y	Y	Y	N/A	N/A
Temat"1" na stronie 667	N/A	N/A	N/A	N/A	Y
TRANSPORT "1" na stronie 667	N/A	Y	N/A	N/A	N/A

Uwaga:

1. Te właściwości nie mają właściwości poziomu aplikacji, ale można je opcjonalnie ustawić za pomocą właściwości administrowanych.

XMS .NET wymagane właściwości dla administrowanych obiektów ConnectionFactory

Gdy aplikacja tworzy fabrykę połączeń, należy zdefiniować pewną liczbę właściwości, aby utworzyć połączenie z serwerem przesyłania komunikatów.

Właściwości wymienione w poniższych tabelach są minimalną wymaganą dla aplikacji, która ma zostać ustawiona w celu utworzenia połączenia z serwerem przesyłania komunikatów. Jeśli chcesz dostosować sposób tworzenia połączenia, aplikacja może w razie potrzeby ustawić dodatkowe właściwości obiektu ConnectionFactory . Więcej informacji na ten temat zawiera sekcja Właściwości elementu ConnectionFactory. Dołączona jest pełna lista dostępnych właściwości.

Połączenie z menedżerem kolejek produktu IBM MQ

Tabela 90. Ustawienia właściwości dla administrowanych obiektów ConnectionFactory dla połączeń z menedżerem kolejek produktu IBM MQ

Wymagana właściwość XMS	**Wymagana jest równoważna właściwość IBM MQ JMS**
XMSC_CONNECTION_TYPE	Produkt XMS działa z tym produktem z właściwości Nazwa klasy fabryki połączeń i TRANSPORT (TRAN).
XMSC_WMQ_HOST_NAME	NAZWA HOSTA (HOST)
PORT XMSC_WMQ_PORT	PORT
MENEDŻER KOLEJEK XMSC_WMQ_QUEUE_MANAGER	Nazwa menedżera kolejek

Połączenie w czasie rzeczywistym z brokerem

Tabela 91. Ustawienia właściwości dla administrowanych obiektów ConnectionFactory dla połączeń w czasie rzeczywistym z brokerem

WymaganeXMS	**Wymagana jest równoważna właściwość IBM MQ JMS**
XMSC_CONNECTION_TYPE	Produkt XMS działa z tym produktem z właściwości Nazwa klasy fabryki połączeń i TRANSPORT (TRAN).
XMSC_RTT_HOST_NAME	NAZWA HOSTA (HOST)
PORT XMSC_RTT_PORT	PORT

Połączenie z WebSphere Application Server service integration bus

Tabela 92. Ustawienia właściwości dla administrowanych obiektów ConnectionFactory dla połączeń z serwerem WebSphere Application Server service integration bus

XMS property	**Opis**
XMSC_CONNECTION_TYPE	Typ serwera przesyłania komunikatów, z którym łączy się aplikacja.. Wartość ta jest 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 właściwości wymagane dla administrowanych obiektów docelowych

Aplikacja tworzący miejsce docelowe musi ustawić kilka właściwości, które aplikacja ma w administrowanego obiekcie docelowym.

Tabela 93. Ustawienia właściwości dla administrowanych obiektów docelowych

Typ połączenia	Właściwość	Opis
IBM MQ menedżer kolejek	KOLEJKA (QU) TOPIC (TOP)	Kolejka, z którą ma zostać nawiązane połączenie Temat używany przez aplikację jako miejsce docelowe
Połączenie w czasie rzeczywistym z brokerem	TOPIC (TOP)	Temat używany przez aplikację jako miejsce docelowe
WebSphere Application Server service integration bus	topicName queueName	Jeśli aplikacja łączy się z tematem Jeśli aplikacja łączy się z kolejką

XMS .NET tworzenie obiektów administrowanych

Definicje obiektów ConnectionFactory i Destination, które są wymagane przez aplikacje produktu XMS w celu nawiązania połączenia z serwerem przesyłania komunikatów, muszą zostać utworzone przy użyciu odpowiednich narzędzi administracyjnych.

Zanim rozpoczniesz

Szczegółowe informacje na temat różnych typów administrowanych repozytorium obiektów obsługiwanych przez produkt XMS zawiera sekcja "Obsługiwane typy repozytorium obiektów administrowanych przez produkt XMS .NET" na stronie 665.

O tym zadaniu

Aby utworzyć administrowane obiekty dla produktu IBM MQ , należy użyć narzędzia administrowania JMS (JMSAdmin) produktu IBM MQ Explorer lub produktu IBM MQ .

Aby utworzyć administrowane obiekty dla produktu IBM MQ lub IBM Integration Bus, należy użyć narzędzia administrowania JMS produktu IBM MQ (JMSAdmin).

Aby utworzyć administrowane obiekty dla produktu WebSphere Application Server service integration bus, należy użyć Konsoli administracyjnej serwera WebSphere Application Server .

V 9.1.2 W narzędziach administracyjnych właściwość jest znana jako **APPLICATIONNAME** lub **APPNAME** w celu skróconej.

Uwaga: Nie można użyć obiektu JMSAdmin do ustawienia transportu (UNMANAGED). Dlatego w celu uzyskania niezarządzanego klienta XMS przy użyciu administracyjnie wybranej nazwy aplikacji należy wprowadzić następującą komendę:

```
cf.SetIntProperty(XMSC.WMQ_CONNECTION_MODE, XMSC.WMQ_CM_CLIENT_UNMANAGED);
```

W poniższych krokach podsumowany jest to, co należy zrobić, aby utworzyć administrowane obiekty.

Procedura

1. Utwórz fabrykę połączeń i zdefiniuj niezbędne właściwości, aby utworzyć połączenie z aplikacji na wybrany serwer.

Właściwości minimalne wymagane przez produkt XMS w celu nawiązania połączenia są zdefiniowane w produkcie "XMS .NET wymagane właściwości dla administrowanych obiektów ConnectionFactory" na stronie 668.

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 celu nawiązania połączenia w czasie rzeczywistym z brokerem należy utworzyć temat.
   - W przypadku połączenia z serwerem WebSphere Application Server service integration busnależy utworzyć kolejkę lub temat.

   Właściwości minimalne wymagane przez produkt XMS w celu nawiązania połączenia są zdefiniowane w produkcie "XMS .NET właściwości wymagane dla administrowanych obiektów docelowych" na stronie 669.

XMS .NET tworzący obiekty InitialContext

Aplikacja musi utworzyć kontekst początkowy, który będzie używany w celu nawiązania połączenia z repozytorium obiektów administrowanych w celu pobrania wymaganych obiektów administrowanych.

O tym zadaniu

Obiekt InitialContext hermetyzuje połączenie z repozytorium. Interfejs API produktu XMS udostępnia metody służące do wykonywania następujących zadań:

- Tworzenie obiektu InitialContext
- Wyszukaj administrowany obiekt w administrowanych repozytorium obiektów.

Procedura

- Więcej szczegółowych informacji na temat tworzenia obiektu InitialContext można znaleźć w sekcji InitialContext dla produktów .NET i Properties of InitialContext.

Właściwości obiektu XMS .NET InitialContext

Parametry konstruktora InitialContext obejmują położenie repozytorium administrowanych obiektów, podane jako jednolity wskaźnik zasobów (URI). Aby aplikacja mogła nawiązać połączenie z repozytorium, może być konieczne podanie większej ilości informacji niż informacje zawarte w identyfikatorze URI.

W interfejsie JNDI i w implementacji .NET produktu XMSinformacje dodatkowe są udostępniane w środowisku Hashtable do konstruktora.

Położenie administrowanego repozytorium obiektów jest zdefiniowane we właściwości XMSC_IC_URL . Ta właściwość jest zwykle przekazywana w wywołaniu Create (tworzenie), ale może być modyfikowana 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 WWW, która korzysta z tych właściwości w celu nawiązania połączenia z katalogiem JNDI.

Następujące właściwości są przekazywane bez modyfikacji do usługi Web Service, która będzie używać ich do łączenia się z katalogiem JNDI.

- URI_DOSTAWCY XMSC_IC_PROVIDER_URL
- XMSC_IC_SECURITY_CREDENTIALS
- XMSC_IC_SECURITY_AUTHENTICATION
- XMSC_IC_SECURITY_PRINCIPAL
- XMSC_IC_SECURITY_PROTOCOL

Format identyfikatora URI dla kontekstów początkowych produktu XMS

Położenie repozytorium administrowanych obiektów jest udostępniane jako jednolity wskaźnik zasobów (URI). Format identyfikatora URI zależy od typu kontekstu.

Kontekst FileSystem

W kontekście FileSystem adres URL określa położenie katalogu opartego na systemie plików. Struktura adresu URL jest zdefiniowana w dokumencie RFC 1738, *Uniform Resource Locators (URL)*: adres URL ma przedrostek `file://`, a składnia następująca 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 używać separatorów '/separatorów lub' \ '.
W przypadku użycia znaku '\', każdy separator musi być poprzedzony znakiem zmiany znaczenia, używając dodatkowego znaku '\'. Zapobiega to próbie interpretowania separatora przez środowisko .NET jako znaku zmiany znaczenia dla tego, co następuje.

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 kontekście LDAP podstawowa struktura adresu URL jest zdefiniowana w dokumencie RFC 2255, *The LDAP URL Format*(Format adresu URL LDAP), z przedrostkiem bez rozróżniania wielkości liter `ldap://` .

Dokładna składnia jest ilustrowana w następującym przykładzie:

```
LDAP://[Hostname][:Port]["/"[DistinguishedName]]
```

Ta składnia jest zdefiniowana w dokumencie RFC, ale bez obsługi żadnych atrybutów, zasięgu, filtrów lub rozszerzeń.

Przykłady tej składni to:

```
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 kontekście WSS adres URL jest w postaci punktu końcowego usług Web Service z przedrostkiem `http://`.

Alternatywnie można użyć przedrostka `cosnaming://` lub `wsvc://`,

Te dwa przedrostki są interpretowane w ten sposób, że używany jest kontekst WSS z adresem URL, do którego dostęp jest uzyskiwany za pośrednictwem protokołu http, co umożliwia łatwe uzyskiwanie początkowego typu kontekstu bezpośrednio z adresu URL.

Przykłady tej składni zawierają następujące elementy:

```
http://madison.ibm.com:9080/xmsjndi/services/JndiLookup
cosnaming://madison/jndilookup
```

Usługa Web Service wyszukiwania JNDI dla produktu XMS .NET

Aby uzyskać dostęp do katalogu nazw COS z poziomu produktu XMS, usługa Web Service wyszukiwania produktu JNDI musi zostać wdrożona na serwerze WebSphere Application Server service integration bus . Ta usługa Web Service tłumaczy informacje o produkcie Java z usługi nazw COS na formularz, który może być odczytany przez aplikacje produktu XMS .

Usługa Web Service jest udostępniana w pliku archiwum korporacyjnego SIBXJndiLookupEAR.ear, który znajduje się w katalogu instalacyjnym. Dla bieżącej wersji produktu IBM Message Service Client for .NETplik SIBXJndiLookupEAR.ear można znaleźć w katalogu *install_dir*\java\lib . Można to zainstalować na serwerze WebSphere Application Server service integration bus przy użyciu Konsoli administracyjnej lub narzędzia skryptowego wsaadmin. Więcej informacji na temat wdrażania aplikacji usług Web Service znajduje się w dokumentacji produktu.

Aby zdefiniować usługę Web Service w aplikacjach produktu XMS , należy po prostu ustawić właściwość XMSC_IC_URL obiektu InitialContext na adres URL punktu końcowego usługi Web Service. Jeśli na przykład usługa Web Service jest wdrażana na hoście serwera o nazwie MyServer, przykładem adresu URL punktu końcowego usługi Web Service jest:

```
wsvc://MyHost:9080/SIBXJndiLookup/services/JndiLookup
```

Ustawienie właściwości XMSC_IC_URL umożliwia wywoływanie funkcji InitialContext Lookup w celu wywołania usługi Web Service w zdefiniowanym punkcie końcowym, co z kolei wyszukuje wymagany administrowany obiekt z usługi nazw COS.

Aplikacje produktu .NET mogą korzystać z usługi Web Service. The server-side deployment is the same for XMS C, /C++ and, XMS .NET. XMS Program .NET wywołuje usługę Web Service bezpośrednio za pomocą Microsoft .NET Framework.

Pobieranie obiektów administrowanych przez program XMS .NET

Program XMS pobiera obiekt administrowany z repozytorium przy użyciu adresu udostępnionego podczas tworzenia obiektu InitialContext lub w właściwościach InitialContext .

Obiekty, które mają zostać pobrane, mogą mieć następujące typy nazw:

- Prosta nazwa opisująca obiekt docelowy, na przykład miejsce docelowe kolejki o nazwie SalesOrders .
- Nazwa złożona, która może być złożona z elementu SubContexts, oddzielona przez '/', i musi kończyć się nazwą obiektu. Przykładem nazwy złożonej jest "Warehouse/PickLists/DispatchQueue2", gdzie Warehouse and Picklists to SubContexts w katalogu nazw, a DispatchQueue2 jest nazwą obiektu docelowego.

Blokowanie aplikacji przy użyciu nowszej wersji produktu XMS

Domyślnie, gdy zainstalowana jest nowsza wersja produktu XMS , aplikacje korzystające z 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 w globalnej pamięci podręcznej zespołu (Global Assembly Cache-GAC) współistnieją wiele instancji podobnych zespołów XMS .NET , ale mają one różne numery wersji. Wewnętrznie, GAC używa pliku strategii do kierowania wywołań aplikacji do najnowszej wersji produktu XMS. Aplikacje działają bez konieczności ponownego kompilowania 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, która składa 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 korzystają z tej samej wersji produktu XMS .NET.

Zabezpieczanie komunikacji dla aplikacji XMS

Ta sekcja zawiera informacje na temat konfigurowania bezpiecznej komunikacji w celu umożliwienia aplikacjom produktu XMS nawiązywania połączeń za pośrednictwem protokołu SSL (Secure Sockets Layer) z mechanizmem przesyłania komunikatów produktu WebSphere Application Server service integration bus lub menedżerem kolejek produktu IBM MQ .

O tym zadaniu

Sekcja zawiera następujące tematy:

- “Bezpieczne połączenia z menedżerem kolejek produktu IBM MQ” na stronie 673
- “Odwzorowania nazw CipherSuite i CipherSpec na potrzeby połączeń produktu XMS z menedżerem kolejek produktu IBM MQ” na stronie 674
- “Bezpieczne połączenia z mechanizmem przesyłania komunikatów produktu WebSphere Application Server service integration bus” na stronie 675
- “Odwzorowania nazw CipherSuite i CipherSpec na potrzeby połączeń z serwerem WebSphere Application Server service integration bus” na stronie 676

Bezpieczne połączenia z menedżerem kolejek produktu IBM MQ

Aby umożliwić aplikacji XMS .NET nawiąże bezpieczne połączenia z menedżerem kolejek produktu IBM MQ , odpowiednie właściwości muszą zostać 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 pakiet CipherSuite jest określony w obiekcie ConnectionFactory .

Jeśli używane są biblioteki klienta IBM WebSphere MQ 7.0.0 Fix Pack 1 i nowsze i nawiązano połączenie z menedżerem kolejek produktu IBM WebSphere MQ 7.0 , można utworzyć wiele połączeń z tym samym menedżerem kolejek w aplikacji XMS . Jednak połączenie z innym menedżerem kolejek nie jest dozwolone. W przypadku próby uzyskania informacji o błędzie `MQRC_SSL_ALREADY_INITIALIZED` .

Jeśli używane są biblioteki klienta IBM WebSphere MQ 6.0 i nowsze, można utworzyć połączenie SSL tylko wtedy, gdy poprzednie połączenie SSL jest zamykane. Wielokrotne współbieżne połączenia SSL z tego samego procesu do tych samych lub różnych menedżerów kolejek nie są dozwolone. W przypadku podjęcia próby wykonania więcej niż jednego żądania, zostanie wyświetlone ostrzeżenie `MQRC_SSL_ALREADY_INITIALIZED`, co może oznaczać, że niektóre żądane parametry połączenia SSL zostały zignorowane.

Właściwości ConnectionFactory dla połączeń za pośrednictwem protokołu SSL do menedżera kolejek produktu IBM MQ z krótkim opisem są przedstawione w poniższej tabeli:

Tabela 94. Właściwości obiektu ConnectionFactory dla połączeń z menedżerem kolejek produktu IBM MQ przy użyciu 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.
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 na potrzeby połączeń produktu XMS z menedżerem kolejek produktu IBM MQ

Element InitialContext tłumaczy się między właściwością fabryki połączeń JMS SSLCIPHERSUITE a XMS odpowiednikiem XMSC_WMQ_SSL_CIPHER_SPEC w pobliżu. Podobne tłumaczenie jest konieczne, jeśli określono wartość dla XMSC_WMQ_SSL_CIPHER_SUITE, ale pominięto wartość dla XMSC_WMQ_SSL_CIPHER_SPEC.

Tabela 95 na stronie 674 zawiera listę dostępnych specyfikacji CipherSpecs i ich odpowiedników JSSE CipherSuite .

Tabela 95. Dostępne odpowiedniki CipherSpecs i ich odpowiedniki 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: Parametr TLS_RSA_WITH_3DES_EDE_CBC_SHA jest nieaktualny. Jednak może być ona nadal używana do przesyłania do 32 GB danych, zanim połączenie zostanie zakończone z błędem AMQ9288. Aby uniknąć tego błędu, należy unikać używania potrójnego algorytmu szyfrowania DES lub włączyć resetowanie klucza tajnego podczas korzystania z tej specyfikacji CipherSpec.

Bezpieczne połączenia z mechanizmem przesyłania komunikatów produktu WebSphere Application Server service integration bus

Aby umożliwić aplikacji XMS .NET nawiąże bezpieczne połączenia z mechanizmem przesyłania komunikatów produktu WebSphere Application Server service integration bus , odpowiednie właściwości muszą zostać zdefiniowane w obiekcie ConnectionFactory .

Produkt XMS udostępnia obsługę protokołu SSL i HTTPS dla połączeń z serwerem WebSphere Application Server service integration bus. Protokół SSL i HTTPS zapewniają bezpieczne połączenia na potrzeby uwierzytelniania i poufności.

Podobnie jak zabezpieczenia produktu WebSphere , zabezpieczenia produktu XMS są konfigurowane w odniesieniu do standardów zabezpieczeń JSSE oraz konwencji 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 element CipherSuite jest określony w obiekcie ConnectionFactory .

Tabela 96 na stronie 675 zawiera listę właściwości, które muszą być zdefiniowane w obiekcie ConnectionFactory .

Tabela 96. Właściwości obiektu ConnectionFactory służące do zabezpieczania 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 znajduje się przykład właściwości ConnectionFactory dla bezpiecznych połączeń z mechanizmem przesyłania komunikatów produktu 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);
```

Jeśli nazwa łańcucha powinna być ustawiona na wartość BootstrapTunneledSecureMessaging lub BootstrapSecureMessaging, a numer_portu to numer portu, na którym serwer startowy nasłuchuje nadchodzących żądań.

Poniżej znajduje się przykład właściwości ConnectionFactory dla bezpiecznych połączeń z mechanizmem przesyłania komunikatów produktu WebSphere Application Server service integration bus z wstawioną przykładową wartością:

```
  /* 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 na potrzeby połączeń z serwerem WebSphere Application Server service integration bus

Ponieważ pakiet 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.

Tabela 97 na stronie 676 zawiera równoważną wartość CipherSpec dla każdego rozpoznanego pakietu CipherSuite.

Tabela 97. Dostępne pakiety CipherSuites i ich odpowiedniki 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
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: Parametr TLS_RSA_WITH_3DES_EDE_CBC_SHA jest nieaktualny. Jednak może być ona nadal używana do przesyłania do 32 GB danych, zanim połączenie zostanie zakończone z błędem AMQ9288. Aby uniknąć tego błędu, należy unikać używania potrójnego algorytmu szyfrowania DES lub włączyć resetowanie klucza tajnego podczas korzystania z tej specyfikacji CipherSpec.

Komunikaty produktu XMS

W tej sekcji opisano strukturę i treść komunikatów produktu XMS oraz wyjaśniono, w jaki sposób aplikacje przetwarzają komunikaty produktu XMS .

Sekcja obejmuje następujące tematy:

- "Części komunikatu produktu XMS" na stronie 676
- "Pola nagłówka w komunikacie XMS" na stronie 677
- "Właściwości komunikatu XMS" na stronie 677
- "Treść komunikatu produktu XMS" na stronie 680
- "Selektory komunikatów" na stronie 684
- "Odwzorowywanie komunikatów produktu XMS na komunikaty produktu IBM MQ" na stronie 684

Części komunikatu produktu 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. Program 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 677.

Zestaw właściwości
Właściwości komunikatu określają dodatkowe informacje na temat komunikatu. 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 677.

Treść
Treść komunikatu zawiera dane aplikacji. Więcej informacji na ten temat zawiera sekcja "Treść komunikatu produktu XMS" na stronie 680.

Aplikacja może wybrać, które komunikaty mają być odbierane. Za pomocą selektorów komunikatów, które określają kryteria wyboru. Kryteria mogą być oparte na wartościach niektórych pól nagłówka i wartościach

dowolnej właściwości komunikatu. Więcej informacji na temat selektorów komunikatów zawiera sekcja “Selektory komunikatów” na stronie 684.

Pola nagłówka w komunikacie XMS

Aby umożliwić aplikacji XMS wymianę komunikatów z aplikacją WebSphere JMS , nagłówek komunikatu XMS zawiera pola nagłówka komunikatu produktu JMS .

Nazwy tych pól nagłówka są rozpoczynanie z przedrostkiem JMS. Opis pól nagłówka komunikatu produktu JMS można znaleźć w sekcji *SpecyfikacjaJava Message Service*.

Produkt XMS implementuje pola nagłówka komunikatu produktu JMS jako atrybuty obiektu komunikatu. Każde pole nagłówka ma własne metody ustawiania i pobierania wartości. Opis tych metod znajduje się w sekcji IMessage. Pole nagłówka jest zawsze czytelne i dostępne do zapisu.

Tabela 98 na stronie 677 zawiera listę pól nagłówka komunikatu produktu JMS i wskazuje, w jaki sposób wartość każdego pola jest ustawiana dla przesłanego komunikatu. Niektóre pola są ustawiane automatycznie przez produkt XMS , gdy aplikacja wysyła komunikat lub, w przypadku obiektu JMSRedelivered, gdy aplikacja odbierze komunikat.

Tabela 98. Pola nagłówka komunikatu produktu JMS.]

Nazwa pola nagłówka komunikatu produktu JMS .	**Sposób ustawiania wartości dla przesłanego komunikatu (w formacie *metoda* [*klasa*])**
JMSCorrelationID	Ustaw wartość JMSCorrelationID [Komunikat]
JMSDeliveryMode	Wyślij [MessageProducer]
JMSDestination	Wyślij [MessageProducer]
JMSExpiration	Wyślij [MessageProducer]
JMSMessageID	Wyślij [MessageProducer]
JMSPriority	Wyślij [MessageProducer]
JMSRedelivered	Receive [MessageConsumer]
JMSReplyTo	Ustaw wartość JMSReplyTo [Komunikat]
JMSTimestamp	Wyślij [MessageProducer]
JMSType	Ustaw atrybut JMSType [Komunikat]

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 program XMS obsługuje następujące predefiniowane właściwości obiektu komunikatu:

- Te same właściwości zdefiniowane przez produkt JMS, które są obsługiwane przez program WebSphere JMS . Nazwy tych właściwości rozpoczynają się od przedrostka JMSX.
- Te same właściwości zdefiniowane przez produkt IBM, które są obsługiwane przez program WebSphere JMS . Nazwy tych właściwości rozpoczynają się od przedrostka JMS_IBM_.

Każda predefiniowana właściwość ma dwie nazwy:

- Nazwa JMS dla zdefiniowanej przez JMSwłaściwości lub nazwy WebSphere JMS dla właściwości zdefiniowanej w produkcie IBM.

  Jest to nazwa, pod którą właściwość jest znana w produkcie JMS lub WebSphere JMS, a także jest nazwą, która jest przesyłana z komunikatem o tej właściwości. Aplikacja XMS używa tej nazwy do identyfikowania właściwości w wyrażeniu selektora komunikatów.

- Nazwa XMS identyfikującą właściwość we wszystkich sytuacjach z wyjątkiem wyrażenia selektora komunikatów. Każda nazwa XMS jest zdefiniowana jako stała nazwana w klasie `IBM.XMS.XMSC` . The value of the named constant is the corresponding JMS or WebSphere JMS name.

Oprócz właściwości predefiniowanych aplikacja XMS może tworzyć i używać własnego zestawu właściwości komunikatu. Te właściwości są nazywane *właściwościami zdefiniowanymi przez aplikację.*

Po utworzeniu komunikatu przez aplikację właściwości komunikatu są czytelne i dostępne do zapisu. Po wysłaniu przez aplikację komunikatu właściwości pozostają czytelne i dostępne do zapisu. Gdy aplikacja odbierze komunikat, właściwości komunikatu są tylko do odczytu. Jeśli aplikacja wywołuje metodę `Clear Properties` klasy Message, gdy właściwości komunikatu są tylko do odczytu, wówczas właściwości stają się czytelne i dostępne do zapisu. Metoda kasuje również właściwości.

Odebrany komunikat po wyczyszczaniu właściwości komunikatu będzie zachowany w sposób zgodny z zachowaniem przekazywania standardowego produktu WMQ XMS dla platformy .NET BytesMessage z wyczyszczonym właściwościami komunikatu.

Jest to jednak niezalecane, ponieważ zostaną utracone następujące właściwości:

- Wartość właściwości JMS_IBM_Encoding, co oznacza, że dane komunikatu nie mogą być zdekodowane w sposób znaczący.
- Wartość właściwości JMS_IBM_Format, co oznacza, że nagłówek łączenia nagłówka (MQMD lub nowy MQRFH2) w nagłówku i istniejących nagłówkach zostanie zerwany.

Aby określić wartości wszystkich właściwości komunikatu, aplikacja może wywołać metodę Get Properties klasy Message. Metoda tworzy iterator, który hermetykuje listę obiektów właściwości, gdzie każdy obiekt właściwości reprezentuje właściwość komunikatu. Aplikacja może następnie użyć metod klasy Iterator w celu pobrania wszystkich obiektów właściwości z kolei, a także użyć metod klasy Property do pobrania nazwy, typu danych i wartości każdej właściwości.

JMS-zdefiniowane właściwości komunikatu

Several JMS-defined properties of a message are supported by both XMS and WebSphere JMS.

Tabela 99 na stronie 678 lists the JMS-defined properties of a message that are supported by both XMS and WebSphere JMS. Opis właściwości zdefiniowanych w produkcie JMSmożna znaleźć w sekcji *Java Message Service Specyfikacja*. Właściwości zdefiniowane w produkcie JMSnie są poprawne dla połączenia w czasie rzeczywistym z brokerem.

W tabeli określa się typ danych każdej właściwości i wskazuje, w jaki sposób wartość właściwości jest ustawiona dla przekazanego komunikatu. Niektóre właściwości są ustawiane automatycznie przez produkt XMS , gdy aplikacja wysyła komunikat lub, w przypadku elementu JMSXDeliveryCount, gdy aplikacja odbiera komunikat.

Tabela 99. JMS-zdefiniowane właściwości komunikatu

Nazwa XMS zdefiniowanej właściwości JMS	**Nazwa JMS**	**Typ danych**	**Sposób ustawiania wartości dla przesłanego komunikatu (w formacie *metoda* [*klasa*])**
JMSX_APPID	JMSXAppID	System.String	Wyślij [MessageProducer]
JMSX_DELIVERY_COUNT	JMSXDeliveryCount	System.Int32	Receive [MessageConsumer]
Identyfikator JMSX_GROUPID	JMSXGroupID	System.String	Ustaw właściwość String [PropertyContext]
JMSX_GROUPSEQ	JMSXGroupSeq	System.Int32	Ustaw właściwość całkowitoliczbową [PropertyContext]
ID_UŻYTKOWNIKA JMSX_USERID	JMSXUserID	System.String	Wyślij [MessageProducer]

IBM-zdefiniowane właściwości komunikatu

Several IBM-defined properties of a message are supported by XMS and WebSphere JMS.

Tabela 100 na stronie 679 lists the IBM defined properties of a message that are supported by both XMS and WebSphere JMS. Więcej informacji na temat właściwości zdefiniowanych w produkcie IBMmożna znaleźć w dokumentacji produktu IBM MQ lub WebSphere Application Server .

W tabeli określa się typ danych każdej właściwości i wskazuje, w jaki sposób wartość właściwości jest ustawiona dla przekazanego komunikatu. Niektóre właściwości są ustawiane automatycznie przez produkt XMS , gdy aplikacja wysyła komunikat.

Tabela 100. Właściwości zdefiniowane przez IBMkomunikatu

Nazwa XMS zdefiniowanej właściwości IBM	**Nazwa WebSphereJMS**	**Typ danych**	**Sposób ustawiania wartości dla przesłanego komunikatu (w formacie *metoda* [*klasa*])**
JMS_IBM_CHARACTER_SET	Zestaw znaków JMS_IBM_Character_Set	System.Int32	Ustaw właściwość całkowitoliczbową [PropertyContext]
JMS_IBM_Encoding	JMS_IBM_Encoding	System.Int32	Ustaw właściwość całkowitoliczbową [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 JMS_IBM_ExceptionProblem	System.String	Receive [MessageConsumer]
JMS_IBM_FEEDBACK	Opinie JMS_IBM_Feedback	System.Int32	Ustaw właściwość całkowitoliczbową [PropertyContext]
FORMAT JMS_IBM_FORMAT	Format JMS_IBM_Format	System.String	Ustaw właściwość String [PropertyContext]
GRUPA_JMS_IBM_LAST_MSG_IN_GROUP	Grupa JMS_IBM_Last_Msg_In_Group	System.Boolean	Ustaw właściwość całkowitoliczbową [PropertyContext]
TYP_JMS_IBM_MSGTYPE	JMS_IBM_MsgType	System.Int32	Ustaw właściwość całkowitoliczbową [PropertyContext]
TYP_JMS_IBM_PUTAPPLTYPE	Typ JMS_IBM_PutAppl	System.Int32	Wyślij [MessageProducer]
DATA JMS_IBM_PUTDATE	JMS_IBM_PutDate	System.String	Wyślij [MessageProducer]
CZAS JMS_IBM_PUTTIME	JMS_IBM_PutTime	System.String	Wyślij [MessageProducer]
JMS_IBM_REPORT_COA	Plik JMS_IBM_Report_COA	System.Int32	Ustaw właściwość całkowitoliczbową [PropertyContext]

Tabela 100. Właściwości zdefiniowane przez IBMkomunikatu (kontynuacja)			
Nazwa XMS zdefiniowanej właściwości IBM	**Nazwa WebSphereJMS**	**Typ danych**	**Sposób ustawiania wartości dla przesłanego komunikatu (w formacie *metoda* [*klasa*])**
JMS_IBM_REPORT_COD	JMS_IBM_Report_COD	System.Int32	Ustaw właściwość całkowitoliczbową [PropertyContext]
JMS_IBM_REPORT_DISCARD_MSG	JMS_IBM_Report_Discard_Msg	System.Int32	Ustaw właściwość całkowitoliczbową [PropertyContext]
JMS_IBM_REPORT_EXCEPTION	Wyjątek JMS_IBM_Report_Exception	System.Int32	Ustaw właściwość całkowitoliczbową [PropertyContext]
JMS_IBM_REPORT_EXPIRATION	Wygaśnięcie JMS_IBM_Report_Expiration	System.Int32	Ustaw właściwość całkowitoliczbową [PropertyContext]
JMS_IBM_REPORT_NAN	Plik JMS_IBM_Report_NAN	System.Int32	Ustaw właściwość całkowitoliczbową [PropertyContext]
JMS_IBM_REPORT_PAN	Plik JMS_IBM_Report_PAN	System.Int32	Ustaw właściwość całkowitoliczbową [PropertyContext]
JMS_IBM_REPORT_PASS_CORREL_ id	JMS_IBM_Report_Pass_Correl_ID	System.Int32	Ustaw właściwość całkowitoliczbową [PropertyContext]
JMS_IBM_REPORT_PASS_MSG_ID	JMS_IBM_Report_Pass_Msg_ID	System.Int32	Ustaw właściwość całkowitoliczbową [PropertyContext]
JMS_IBM_SYSTEM_MESSAGEID	JMS_IBM_System_MessageID	System.String	Wyślij [MessageProducer]

Właściwości komunikatu zdefiniowane przez aplikację

Aplikacja XMS może tworzyć i używać własnego zestawu właściwości komunikatu. Gdy aplikacja wysyła komunikat, te właściwości są również przesyłane razem z komunikatem. Aplikacja odbierający, korzystając z selektorów komunikatów, może następnie wybrać komunikaty, które mają być odbierane w oparciu o wartości tych właściwości.

Aby umożliwić aplikacji WebSphere JMS wybór 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 138. Wartość właściwości definiowanej 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 produktu XMS

Treść komunikatu zawiera dane aplikacji. Jednak komunikat nie może mieć treści i zawierać tylko pola nagłówka i właściwości.

Produkt XMS obsługuje pięć typów treści komunikatu:

Bajty
Treść zawiera strumień bajtów. Komunikat z tym typem treści jest nazywany *komunikatem bajtów*. Interfejs IBytesMessage zawiera metody przetwarzania treści komunikatu bajtów.

Odwzoruj
Treść zawiera zestaw par nazwa-wartość, w których każda wartość ma powiązany typ danych. Komunikat z tym typem treści jest nazywany *komunikatem mapy*. Interfejs IMapMessage zawiera metody przetwarzania treści komunikatu mapy.

Obiekt
Treść zawiera serializowany obiekt Java lub .NET . Komunikat z tym typem treści jest nazywany *komunikatem obiektu*. Interfejs IObjectMessage zawiera metody przetwarzania treści komunikatu obiektu.

Strumień
Treść zawiera strumień wartości, w którym każda wartość ma powiązany typ danych. Komunikat z tym typem treści jest nazywany *komunikatem strumienia*. Interfejs IStreamMessage zawiera metody przetwarzania treści komunikatu strumienia.

Tekstowy
Treść zawiera łańcuch. Komunikat z tym typem treści jest nazywany *komunikatem tekstowym*. Interfejs ITextMessage zawiera metody przetwarzania treści komunikatu tekstowego.

Interfejs IMessage jest nadrzędny względem wszystkich obiektów komunikatów i może być używany w funkcjach przesyłania komunikatów do reprezentowania dowolnego typu komunikatu produktu XMS .

Więcej informacji na temat wielkości oraz wartości maksymalnych i minimalnych dla każdego z tych typów danych zawiera sekcja Tabela 85 na stronie 656.

Komunikaty bajtowe

Treść komunikatu bajtowego zawiera strumień bajtów. Jednostka zawiera tylko rzeczywiste dane, a odpowiedzialność za ich interpretowanie i odbieranie polega na wysyłaniu i odbierającym.

Komunikaty bajtowe są przydatne, jeśli aplikacja produktu XMS wymaga wymiany komunikatów z aplikacjami, które nie korzystają z interfejsu programistycznego aplikacji XMS lub JMS .

Po utworzeniu przez aplikację komunikatu bajtowego, treść komunikatu jest tylko do zapisu. Aplikacja zestawia dane aplikacji do treści, wywołując odpowiednie metody zapisu w interfejsie IBytesMessage dla produktu .NET. Za każdym razem, gdy aplikacja zapisuje wartość do strumienia komunikatów bajtów, wartość ta jest składana natychmiast po poprzedniej wartości zapisanej przez aplikację. XMS utrzymuje kursor wewnętrzny, aby pamiętać o pozycji ostatniego bajtu, który został zmontowany.

Po wysłaniu komunikatu przez aplikację treść komunikatu staje się dostępna tylko do odczytu. W tym trybie aplikacja może wielokrotnie wysyłać komunikat.

Gdy aplikacja odbierze komunikat w postaci bajtów, treść komunikatu jest tylko do odczytu. Aplikacja może użyć odpowiednich metod odczytu interfejsu IBytesMessage , aby odczytać zawartość strumienia komunikatów bajtów. Aplikacja odczytuje bajty w sekwencji, a program XMS utrzymuje kursor wewnętrzny, aby zapamiętać pozycję ostatniego odczytanego bajtu.

Jeśli aplikacja wywoła metodę Reset interfejsu IBytesMessage , gdy treść komunikatu bajtowego jest dostępna do zapisu, treść staje się dostępna tylko do odczytu. Metoda ta umożliwia również ponowne pozycjonowanie kursora na początku strumienia komunikatów.

Jeśli aplikacja wywoła metodę `Clear Body` interfejsu IMessage dla .NET , gdy treść komunikatu bajtowego jest tylko do odczytu, wówczas treść staje się dostępna do zapisu. Metoda również czyści ciało.

Komunikaty mapy

Treść komunikatu mapy zawiera zestaw par nazwa-wartość, w których każda wartość ma powiązany typ danych.

W każdej parze nazwa-wartość nazwa jest łańcuchem, który identyfikuje wartość, a wartość jest elementem danych aplikacji, które mają jeden z typów danych XMS wymienionych w Tabela 101 na stronie 683. Kolejność par nazwa-wartość nie jest zdefiniowana. Klasa MapMessage zawiera metody służące do ustawiania i pobierania par nazwa-wartość.

Aplikacja może losowo uzyskać dostęp do pary nazwa-wartość, określając jej nazwę.

Aplikacja .NET może użyć właściwości MapNames , aby uzyskać wyliczenie nazw w treści komunikatu mapy.

Jeśli aplikacja pobiera wartość pary nazwa-wartość, wartość może zostać przekształcona przez program XMS na 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 “Niejawne przekształcenie wartości właściwości z jednego typu danych na inny.” na stronie 656.

Po utworzeniu przez aplikację komunikatu odwzorowania treść komunikatu jest czytelna i dostępna do zapisu. Po wysłaniu przez aplikację wiadomości treść pozostaje czytelna i dostępna do zapisu. Gdy aplikacja odbierze komunikat mapy, treść komunikatu jest tylko do odczytu. Jeśli aplikacja wywoła metodę Clear Body klasy Message, gdy treść komunikatu mapy jest tylko do odczytu, wówczas treść staje się czytelna i dostępna do zapisu. Metoda również czyści ciało.

Komunikaty obiektu

Treść komunikatu obiektu zawiera serializowany obiektJava lub .NET .

Aplikacja XMS może odbierać komunikat obiektu, zmieniać jego pola nagłówka i właściwości, a następnie wysyłać je do innego miejsca docelowego. Aplikacja może również skopiować treść komunikatu obiektu i użyć go do utworzenia innego komunikatu obiektu. Program XMS traktuje treść komunikatu obiektu w postaci tablicy bajtów.

Po utworzeniu przez aplikację komunikatu obiektu, treść komunikatu jest czytelna i dostępna do zapisu. Po wysłaniu przez aplikację wiadomości treść pozostaje czytelna i dostępna do zapisu. Gdy aplikacja odbierze komunikat o obiekcie, treść komunikatu jest tylko do odczytu. Jeśli aplikacja wywoła metodę `Clear Body` interfejsu IMessage dla .NET , gdy treść komunikatu obiektu jest tylko do odczytu, wówczas treść staje się czytelna i dostępna do zapisu. Metoda również czyści ciało.

Komunikaty strumienia

Treść komunikatu strumienia zawiera strumień wartości, w którym każda wartość ma powiązany typ danych.

Typ danych wartości to jeden z typów danych produktu XMS wymienionych w sekcji Tabela 101 na stronie 683.

Po utworzeniu przez aplikację komunikatu strumienia treść komunikatu jest zapisywalny. Aplikacja zestawia dane aplikacji do treści, wywołując odpowiednie metody zapisu w interfejsie IStreamMessage dla produktu .NET. Za każdym razem, gdy aplikacja zapisuje wartość do strumienia komunikatów, wartość, a jej typ danych są składane natychmiast po poprzedniej wartości zapisanej przez aplikację. XMS utrzymuje kursor wewnętrzny, aby pamiętać o pozycji ostatniej zmontowanej wartości.

Po wysłaniu komunikatu przez aplikację treść komunikatu staje się dostępna tylko do odczytu. W tym trybie aplikacja może wysłać komunikat wiele razy.

Gdy aplikacja odbierze komunikat strumienia, treść komunikatu jest tylko do odczytu. Aplikacja może użyć odpowiednich metod odczytu interfejsu IStreamMessage dla produktu .NET w celu odczytania treści strumienia komunikatów. Aplikacja odczyta wartości w sekwencji, a program XMS utrzymuje kursor wewnętrzny, aby pamiętać o pozycji ostatniej odczytanej wartości.

Gdy aplikacja odczytuje wartość ze strumienia komunikatów, wartość może zostać przekształcona przez program XMS na 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 “Niejawne przekształcenie wartości właściwości z jednego typu danych na inny.” na stronie 656.

Jeśli błąd wystąpi podczas próby odczytania wartości ze strumienia komunikatów przez aplikację, kursor nie jest zaawansowany. Aplikacja może wykonać odtwarzanie po wystąpieniu błędu, próbując odczytać wartość jako inny typ danych.

Jeśli aplikacja wywoła metodę Reset interfejsu IStreamMessage dla XMS , gdy treść komunikatu strumienia jest tylko do zapisu, wówczas treść staje się tylko do odczytu. Metoda ta umożliwia również ponowne pozycjonowanie kursora na początku strumienia komunikatów.

Jeśli aplikacja wywoła metodę `Clear Body` interfejsu IMessage dla XMS , gdy treść komunikatu strumienia jest tylko do odczytu, wówczas treść staje się tylko do zapisu. Metoda również czyści ciało.

Komunikaty tekstowe

Treść wiadomości tekstowej zawiera łańcuch.

Po utworzeniu przez aplikację wiadomości tekstowej treść wiadomości jest czytelna i dostępna do zapisu. Po wysłaniu przez aplikację wiadomości treść pozostaje czytelna i dostępna do zapisu. Gdy aplikacja otrzymuje wiadomość tekstową, treść wiadomości jest tylko do odczytu. Jeśli aplikacja wywoła metodę Clear Body interfejsu IMessage dla programu .NET , gdy treść komunikatu tekstowego jest tylko do odczytu, wówczas treść staje się czytelna i dostępna do zapisu. Metoda również czyści ciało.

Typy danych dla elementów danych aplikacji

Aby aplikacja XMS mogła wymieniać komunikaty z aplikacją IBM MQ classes for JMS , zarówno aplikacje, jak i aplikacje muszą być w stanie interpretować dane aplikacji w treści komunikatu w ten 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 101 na stronie 683. Dla każdego typu danych w tabeli wyświetlany jest zgodny typ danych Java . Produkt XMS udostępnia metody do zapisywania elementów danych aplikacji tylko z tymi typami danych.

Tabela 101. Typy danych produktu XMS zgodne z typami danych produktu 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 podpisanych zmiennopozycyjnych	liczba zmiennopozycyjna
System.Double	Liczba zmiennopozycyjna o podwójnej precyzji podpisanej	double (podwójna)
System.String	Łańcuch znaków	Łańcuch

Więcej informacji na temat wielkości, wartości maksymalnej i minimalnej wartości każdego z tych typów danych zawiera sekcja “Typy podstawowe produktu XMS” na stronie 656.

Selektory komunikatów

Aplikacja XMS używa selektorów komunikatów w celu wybrania komunikatów, które mają zostać odebrane.

Gdy aplikacja tworzy konsument komunikatów, może powiązać wyrażenie selektora komunikatów z konsumentem. Wyrażenie selektora komunikatów określa kryteria wyboru.

Gdy aplikacja łączy się z menedżerem kolejek produktu IBM WebSphere MQ 7.0 , wybór komunikatów jest wybierany po stronie menedżera kolejek. Produkt XMS nie wykonuje żadnego wyboru i po prostu dostarcza komunikat, który otrzymał od menedżera kolejek, co zapewnia lepszą wydajność.

Aplikacja może utworzyć więcej niż jeden konsument komunikatów, każdy z własnym wyrażeniem 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 zdefiniowane przez aplikację

Może on 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 komunikatów z tym wyrażeniem selektora komunikatów tylko wtedy, gdy ma on priorytet większy niż 3; właściwość zdefiniowana przez aplikację, producent, o wartości `Jaguar;` i innej zdefiniowanej właściwości aplikacji, modelu z wartością `xj6` lub `xj12.` .

Reguły składni służące do tworzenia wyrażenia selektora komunikatów w programie XMS są takie same, jak w przypadku produktu IBM MQ classes for JMS. Informacje na temat konstruowania wyrażenia selektora komunikatów można znaleźć w dokumentacji produktu IBM MQ . Należy zwrócić uwagę, że w wyrażeniu selektora komunikatów nazwy właściwości zdefiniowanych przez produkt JMSmuszą być nazwami JMS, a nazwy właściwości zdefiniowanych w produkcie IBMmuszą być nazwami IBM MQ classes for JMS . Nie można używać nazw XMS w wyrażeniu selektora komunikatów.

Odwzorowywanie komunikatów produktu XMS na komunikaty produktu IBM MQ

Pola nagłówka JMS i właściwości komunikatu produktu XMS są odwzorowywane na pola w strukturach nagłówka komunikatu produktu 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, w jaki 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 w komunikacie IBM MQ . Ustawienie w ten sposób właściwości XMSC_WMQ_TARGET_CLIENT zakłada, że aplikacja, która odbiera komunikat, może obsługiwać nagłówek MQRFH2 . Aplikacją odbierającą może być więc inna aplikacja XMS , aplikacja IBM MQ classes for JMS lub rodzima aplikacja IBM MQ , która została zaprojektowana do obsługi nagłówka MQRFH2 .

Jeśli właściwość XMSC_WMQ_TARGET_CLIENT obiektu docelowego jest ustawiona na wartość XMSC_WMQ_TARGET_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 nagłówka JMS i właściwości, które nie mogą zostać odwzorowane na pola w strukturze nagłówka MQMD, są ignorowane. Aplikacja, która odbiera komunikat, może być więc rodzimą IBM MQ , która nie jest przeznaczona do obsługi nagłówka MQRFH2 .

Komunikaty produktu IBM MQ odebrane z menedżera kolejek są odwzorowywane na komunikaty produktu XMS w taki sam sposób, w jaki 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 przez wartość właściwości **Msd** zawartej w folderze `mcd` nagłówka MQRFH2 . Jeśli właściwość **Msd** nie znajduje się w nagłówku MQRFH2 lub jeśli komunikat IBM MQ nie ma nagłówka MQRFH2 , wynikowy komunikat XMS będzie miał treść, której typ jest określany na podstawie wartości 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 bajtów. Jeśli w komunikacie IBM MQ nie ma nagłówka MQRFH2 , ustawiane są tylko te pola nagłówka JMS i właściwości, które mogą pochodzić z pól w nagłówku MQMD.

Więcej informacji na temat odwzorowywania komunikatów programu IBM MQ classes for JMS na komunikaty produktu IBM MQ zawiera sekcja “Odwzorowywanie komunikatów produktu JMS na komunikaty produktu IBM MQ” na stronie 142.

Odczytywanie i zapisywanie deskryptora komunikatu z aplikacji produktu IBM Message Service Client for .NET

Można uzyskać dostęp do wszystkich pól deskryptora komunikatu (MQMD) komunikatu IBM MQ z wyjątkiem pól StrucId i Version; BackoutCount można odczytać, ale nie zapisywać do niego. Ta funkcja jest dostępna tylko wtedy, gdy nawiąże połączenie z menedżerem kolejek produktu IBM WebSphere MQ 6.0 lub nowszego i jest kontrolowana przez właściwości miejsca docelowego opisane w dalszej części.

Atrybuty komunikatów udostępniane przez produkt IBM Message Service Client for .NET ułatwiają aplikacjom produktu XMS ustawianie pól MQMD, a także na potrzeby obsługi aplikacji IBM WebSphere MQ .

Niektóre ograniczenia mają zastosowanie podczas przesyłania komunikatów w trybie publikowania/subskrypcji. Na przykład, pola MQMD, takie jak MsgID i CorrelId, jeśli są ustawione, są ignorowane.

Funkcja opisana w tym temacie jest niedostępna w przypadku przesyłania komunikatów w trybie publikowania/subskrypcji podczas nawiązywania połączenia z menedżerem kolejek produktu IBM WebSphere MQ 6.0 . Jest ona również niedostępna, gdy właściwość **PROVIDERVERSION** jest ustawiona na wartość 6.

Uzyskiwanie dostępu do danych komunikatu produktu IBM MQ z aplikacji IBM Message Service Client for .NET

Można uzyskać dostęp do pełnych danych komunikatu produktu IBM MQ , w tym nagłówka MQRFH2 (jeśli istnieje) i innych nagłówków produktu IBM MQ (jeśli istnieją) w aplikacji produktu IBM Message Service Client for .NET jako treści elementu JMSBytesMessage.

Funkcja opisana w tym temacie jest dostępna tylko wtedy, gdy nawiąże połączenie z menedżerem kolejek produktu IBM WebSphere MQ 7.0 lub późniejszym, a 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 nagłówek MQRFH2 , jeśli jest obecny) jako treść komunikatu JMSBytesMessage.

Korzystanie z interfejsu modelu obiektu komponentu (klasy automatyzacji produktuIBM MQ dla elementu ActiveX)

Klasy automatyzacji produktu IBM MQ dla elementów ActiveX (MQAX) to komponenty ActiveX , które udostępniają klasy, których można używać w aplikacji w celu uzyskania dostępu do produktu IBM MQ.

W produkcie IBM MQ 9.0obsługa produktu IBM MQ dla produktu Microsoft Active X jest nieaktualna. Zalecaną technologią zastępczą są klasy IBM MQ dla .NET. Więcej informacji na ten temat zawiera sekcja Tworzenie aplikacji .NET.

Produkt MQAX wymaga środowiska IBM MQ i odpowiedniej aplikacji produktu IBM MQ , z którą ma być komunikowana.

Dzięki temu aplikacja ActiveX może uruchamiać transakcje i uzyskiwać dostęp do danych w dowolnym systemie korporacyjnym, do którego można uzyskać dostęp za pośrednictwem produktu IBM MQ.

Klasy automatyzacji produktu IBM MQ dla elementu ActiveX:

- Dostęp do funkcji i funkcji interfejsu API produktu IBM MQ można uzyskać, zezwalając na pełne połączenie wzajemne z innymi platformami produktu IBM MQ .
- Zgodne z normalnymi konwencjami, które są oczekiwane w przypadku komponentu ActiveX .
- Jest on zgodny z modelem obiektu IBM MQ , który jest również dostępny dla produktów .NET, C + +, Javai LotusScript.

Dostępne są przykłady startowe MQAX. Tych przykładów można użyć początkowo w celu sprawdzenia, czy instalacja produktu MQAX powiodła się i czy w lokalizacji istnieje podstawowe środowisko produktu IBM MQ . Przykłady pokazują również, w jaki sposób można użyć produktu MQAX.

Skrypty COM i ActiveX

Model obiektu komponentu (Component Object Model-COM) jest modelem programistycznym opartym na obiektowym systemie zdefiniowanym przez produkt Microsoft. Określa on, w jaki sposób komponenty oprogramowania mogą być udostępniane w sposób umożliwiający ich zlokalizowanie i komunikację niezależnie od języka komputera, w którym są one napisane lub w którym znajdują się ich położenie.

ActiveX to zestaw technologii opartych na technologii COM, który integruje projektowanie aplikacji, komponenty wielokrotnego użytku oraz technologie internetowe na platformach Microsoft Windows . Komponenty ActiveX udostępniają interfejsy, które mogą być dostępne dynamicznie przez aplikacje. Klient skryptowy ActiveX to aplikacja, na przykład kompilator, która może zbudować lub wykonać program lub skrypt, który korzysta z interfejsów udostępnianych przez komponenty ActiveX (lub COM).

Obsługa środowiska IBM MQ

Klasy automatyzacji produktu IBM MQ dla elementu ActiveX mogą być używane tylko z klientami **32-bitowych** skryptów ActiveX .

Komponent COM może być używany tylko dla aplikacji **32-bitowych** . Aby napisać 64-bitową aplikację COM, można użyć interfejsu .NET .

Aby uruchomić program MQAX w środowisku serwera IBM MQ , w systemie musi być zainstalowany produkt Windows 2000 lub nowszy.

Aby uruchomić program MQAX w środowisku IBM MQ MQI client , należy w systemie Windows 2000 lub nowszym zainstalować produkt IBM MQ MQI client w systemie:

Serwer IBM MQ MQI client wymaga dostępu do co najmniej jednego serwera IBM MQ . Jeśli w systemie są zainstalowane zarówno serwer IBM MQ MQI client , jak i serwer IBM MQ , aplikacje MQAX zawsze działają na serwerze. Interfejs ActiveX do interfejsu MQAI jest dostępny tylko w środowiskach serwerów IBM MQ .

Projektowanie i programowanie za pomocą klas automatyzacji IBM MQ dla ActiveX

Projektowanie aplikacji MQAX, które uzyskują dostęp do aplikacji innych niżActiveX

Klasy automatyzacji produktu IBM MQ zapewniają dostęp do funkcji interfejsu API produktu IBM MQ . Z tego powodu wszystkie korzyści, które mogą być używane przez produkt IBM MQ , mogą być korzystne dla aplikacji Windows .

Ogólna konstrukcja aplikacji jest taka sama, jak w przypadku dowolnej aplikacji produktu IBM MQ , dlatego należy wziąć pod uwagę wszystkie aspekty projektu opisane w sekcji “Uwagi dotyczące projektowania aplikacji produktu IBM MQ” na stronie 50 .

Aby korzystać z klas automatyzacji produktu IBM MQ , należy zakodować programy Windows w aplikacji przy użyciu języka, który obsługuje tworzenie i używanie obiektów COM. Na przykład Visual Basic, Javai inne klienty skryptowe ActiveX . Klasy mogą być następnie łatwo zintegrowane z aplikacją, ponieważ potrzebne obiekty produktu IBM MQ mogą być kodowane przy użyciu rodzimej składni języka implementacji.

Korzystanie z klas automatyzacji produktu IBM MQ dla elementu ActiveX

Podczas projektowania aplikacji ActiveX , która korzysta z klas automatyzacji IBM MQ dla ActiveX, najważniejszym elementem informacji jest komunikat wysyłany lub odbierany ze zdalnego systemu IBM MQ . Dlatego należy znać format elementów, które są wstawiane do komunikatu. W przypadku skryptu MQAX do pracy, zarówno ten, jak i aplikacja IBM MQ , która pobiera lub wysyła komunikat, musi znać strukturę komunikatu.

Jeśli wysyłany jest komunikat z aplikacją MQAX i ma zostać wykonana konwersja danych na końcu MQAX, należy również wiedzieć, że:

- Strona kodowa używana przez system zdalny
- Kodowanie używane przez system zdalny

Aby zachować przenośny kod, zaleca się ustawienie strony kodowej i kodowania, nawet jeśli są one takie same w systemach wysyłających i odbierających.

Rozważając strukturę implementacji projektowanego systemu, należy pamiętać, że skrypty MQAX działają na tym samym komputerze, na którym znajduje się menedżer kolejek produktu IBM MQ lub klient IBM MQ .

Wskazówki dotyczące programowania

Następujące wskazówki i wskazówki nie są w znaczącym porządku. Są to tematy, które, jeśli mają znaczenie dla pracy, którą wykonujesz, mogą zaoszczędzić czas.

Właściwości deskryptora komunikatu

W przypadku manipulowania właściwościami deskryptora komunikatu w programie może być lepiej użycie szesnastkowych odpowiedników pól.

Informacje zawarte w tej sekcji odnoszą się do następujących właściwości:

- AccountingToken
- CorrelationId
- GroupId
- MessageId

Jeśli aplikacja IBM MQ jest inicjatorem komunikatu, a produkt IBM MQ generuje te właściwości, lepiej jest użyć właściwości AccountingTokenHex, CorrelationIdHex, GroupIdHex i MessageIdHex, jeśli mają być

wyświetlane ich wartości lub manipulować nimi w dowolny sposób, w tym przekazywanie ich z powrotem do programu IBM MQ. Wynika to z tego, że wygenerowane wartości IBM MQ są łańcuchami bajtów, które mają dowolną wartość z zakresu od 0 do 255 włącznie, nie są łańcuchami znaków drukowalnych.

W przypadku, gdy skrypt MQAX jest inicjatorem komunikatu, można użyć właściwości AccountingToken, CorrelationId, GroupIdi MessageId lub ich odpowiedników Hex.

IBM MQ stałe

Stałe IBM MQ są udostępniane jako elementy typu wyliczeniowego IBM MQ w bibliotece MQAX200.

Stałe łańcuchowe IBM MQ

Stałe łańcuchowe produktu IBM MQ nie są dostępne, jeśli używane są klasy automatyzacji produktu IBM MQ dla elementu ActiveX. Należy użyć jawnego łańcucha znaków dla tych, które znajdują się na poniższej liście, a także innych, które mogą być potrzebne. Komendy muszą być dopełniane do ośmiu znaków za pomocą spacji:

Tabela 102. IBM MQ stałe łańcuchowe i odpowiadające im łańcuchy znaków.

Stała łańcuchowa	**Odpowiedni łańcuch znaków**
MQFMT_NONE	" "
ADMINISTRATOR MQFMT_ADMIN	"MQADMIN"
MQFMT_CHANNEL_COMPLETED	"MQCHCOM"
MQFMT_CICS	"MQCICS"
MQFMT_COMMAND_1	"MQCMD1 "
MQFMT_COMMAND_2	"MQCMD2 "
MQFMT_DEAD_LETTER_HEADER	"MQDEAD"
MQFMT_DIST_HEADER	"MQHDIST"
Zdarzenie MQFMT_EVENT	"MQEVENT"
MQFMT_IMS	"MQIMS"
MQFMT_IMS_VAR_STRING	"MQIMSVS"
MQFMT_MD_EXTENSION	"MQHMDE"
MQFMT_PCF	"MQPCF"
MQFMT_REF_MSG_HEADER	"MQHREF"
MQFMT_RF_HEADER	"MQHRF"
MQFMT_STRING	"MQSTR"
MQFMT_TRIGGER	"MQTRIG"
Nagłówek MQFMT_WORK_INFO_HEADER	"MQHWIH"
MQFMT_XMIT_Q_HEADER	"MQXMIT"

Stałe łańcuchowe o wartości NULL

Stałe IBM MQ używane do inicjowania czterech właściwości MQMessage, MQMI_NONE (24 znaki NULL), MQCI_NONE (24 NULL), MQGI_NONE (24 NULL) i MQACT_NONE (32 znaki NULL) nie są obsługiwane przez klasy automatyzacji produktu IBM MQ dla elementu ActiveX. Ustawienie ich na puste łańcuchy ma taki sam efekt.

Na przykład, aby ustawić różne identyfikatory komunikatu MQMessage na następujące wartości: *mymessage*. **MessageId** = "" *mymessage*. **CorrelationId** = "" *mymessage*. **AccountingToken** = ""

Odbieranie komunikatu z programu IBM MQ

Istnieje kilka sposobów otrzymywania komunikatu z produktu IBM MQ:

- Odpytywanie przez wydanie komendy GET, po której następuje oczekiwanie, przy użyciu funkcji Visual Basic TIMER.
- Wydanie komendy GET z opcją oczekiwania; należy określić przedział czasu oczekiwania, ustawiając właściwość WaitInterval . Należy wziąć pod uwagę, że nawet jeśli system jest uruchamiany w środowisku wielowątkowym, wówczas oprogramowanie działające w danym momencie może działać tylko w postaci jednowątkowej. Pozwala to na uniknięcie blokady systemu w nieskończoność.

  Nie ma wpływu na inne wątki. Jeśli jednak inne wątki wymagają dostępu do produktu IBM MQ, wymagają one drugiego połączenia z produktem IBM MQ przy użyciu dodatkowych menedżerów kolejek i obiektów kolejki MQAX.

Wydanie komendy GET z opcją oczekiwania i ustawienie parametru WaitInterval na wartość MQWI_UNLIMITED powoduje, że system zablokuje się do momentu zakończenia operacji GET, jeśli proces jest jednowątkowy.

Używanie konwersji danych

Dwie formy konwersji danych są obsługiwane przez klasy automatyzacji produktu IBM MQ dla ActiveX -kodowanie liczbowe i konwersja zestawu znaków.

Kodowanie liczbowe

Jeśli właściwość Kodowanie komunikatu MQMessage zostanie ustawiona, następujące metody są przekształcane między różnymi numerycznymi systemami kodowania:

- Metoda ReadDecimal2
- Metoda ReadDecimal4
- Metoda ReadDouble
- Metoda ReadDouble4
- Metoda ReadFloat
- Metoda ReadInt2
- Metoda ReadInt4
- Metoda ReadLong
- Metoda ReadShort
- Metoda ReadUInt2
- Metoda WriteDecimal2
- Metoda WriteDecimal4
- Metoda WriteDouble
- Metoda WriteDouble4
- Metoda WriteFloat
- Metoda WriteInt2
- Metoda WriteInt4
- Metoda WriteLong
- Metoda WriteShort
- Metoda WriteUInt2

Właściwość Kodowanie może być ustawiona i interpretowana przy użyciu podanych stałych IBM MQ . Rysunek 60 na stronie 690 przedstawia przykład następujących elementów:

```
/* Encodings for Binary Integers */
MQENC_INTEGER_UNDEFINED
MQENC_INTEGER_NORMAL
MQENC_INTEGER_REVERSED

/* Encodings for Decimals */
MQENC_DECIMAL_UNDEFINED
MQENC_DECIMAL_NORMAL
MQENC_DECIMAL_REVERSED

/* Encodings for Floating-Point Numbers */
MQENC_FLOAT_UNDEFINED
MQENC_FLOAT_IEEE_NORMAL
MQENC_FLOAT_IEEE_REVERSED
MQENC_FLOAT_S390
```

Rysunek 60. Dostarczone stałe IBM MQ do kodowania

Na przykład, aby wysłać liczbę całkowitą z systemu Intel do systemu operacyjnego System/390 w kodowaniu System/390 :

```
Dim msg As New MQMessage  'Define an IBM MQ message for our use..
Print msg. Encoding   'Currently 546 (or X'222')
                                          'Set the encoding property
                                           to 785 (or X'311')
msg. Encoding = MQENC_INTEGER_NORMAL OR MQENC_DECIMAL_NORMAL
                        OR MQENC_FLOAT_S390
Print msg. Encoding   'Print it to see the change
Dim local_num As long 'Define a long integer
local_num = 1234                          'Set it
msg. WriteLong (local_num)       'Write the number into the message
```

Konwersja zestawu znaków

Konwersja zestawu znaków jest niezbędna, gdy wysyłany jest komunikat z jednego systemu do innego systemu, w którym strony kodowe są różne. Konwersja strony kodowej jest używana przez:

- Metoda ReadString
- Metoda ReadNullTerminatedString
- Metoda WriteString
- Metoda WriteNullTerminatedString
- Właściwość MessageData

Właściwość MQMessage CharacterSet należy ustawić na obsługiwaną wartość zestawu znaków (CCSID).

Klasy automatyzacji IBM MQ dla ActiveX korzystają z tabel konwersji w celu wykonania konwersji zestawu znaków.

Na przykład, aby automatycznie przekształcić łańcuchy w stronę kodową 437:

```
Dim msg As New MQMessage                 'Define an IBM MQ message
msg.CharacterSet = 437                 'Set code page required
msg.WriteString "A character string"'Put character string in message
```

Metoda WriteString odbiera dane łańcuchowe (`A character string` w tym przykładzie) jako łańcuch Unicode. Następnie przekształca te dane z kodu Unicode na stronę kodową 437 przy użyciu tabeli konwersji 34B001B5.TBL.

Znaki w łańcuchu Unicode, które nie są obsługiwane przez stronę kodową 437, otrzymują standardowy znak zastępczy ze strony kodowej 437.

Podobnie, jeśli używana jest metoda ReadString , komunikat przychodzący ma zestaw znaków ustalony przez wartość deskryptora komunikatu produktu IBM MQ (MQMD), a przed przesłaniem go do języka skryptowego istnieje konwersja z tej strony kodowej na Unicode.

Wątki

Klasy automatyzacji produktu IBM MQ dla elementu ActiveX implementują model o swobodnej wielowątkowości, w którym obiekty mogą być używane między wątkami.

Podczas gdy produkt MQAX zezwala na użycie obiektów MQQueue i MQQueueManager , produkt IBM MQ nie zezwala obecnie na współużytkowanie uchwytów między różnymi wątkami.

Próby użycia tych elementów w innym wątku powodują wystąpienie błędu, a program IBM MQ zwraca kod powrotu MQRC_HCONN_ERROR.

Uwaga: Dla każdego procesu istnieje tylko jeden obiekt MQSession. Użycie funkcji MQSession CompletionCode i ReasonCode nie jest zalecane w środowiskach wielowątkowych. Wartości błędów MQSession mogą zostać nadpisane przez drugi wątek między błędem podniesionym i sprawdzonym w pierwszym wątku. Wątki są serializowane przez czas trwania każdego wywołania metody lub dostępu do właściwości. Dlatego wydanie komendy Get z opcją oczekiwania powoduje, że inne wątki uzyskają dostęp do obiektów MQAX, które mają zostać zawieszone do czasu zakończenia operacji.

Obsługa błędów

Te informacje opisują właściwości obiektu MQAX, sposób obsługi błędów, reguły opisujące sposób obsługi wyjątków i uzyskiwanie właściwości.

Każdy obiekt MQAX zawiera właściwości służące do przechowywania informacji o błędach i metody ich resetowania lub czyszczenia. Dostępne są następujące właściwości:

- CompletionCode
- ReasonCode
- ReasonName

Metoda jest następująca:

- Kody ClearError

Jak działa błąd

Skrypt lub aplikacja MQAX wywołuje metodę obiektu MQAX lub uzyskuje dostęp do niej lub aktualizuje właściwość obiektu MQAX:

1. Wartości ReasonCode i CompletionCode w danym obiekcie są aktualizowane.
2. Atrybuty ReasonCode i CompletionCode w obiekcie MQSession są również aktualizowane przy użyciu tych samych informacji.

   Uwaga: Sekcja “Wątki” na stronie 691 zawiera ograniczenia dotyczące używania kodów błędów MQSession w aplikacjach wielowątkowych.

   Jeśli właściwość CompletionCode jest większa lub równa właściwości ExceptionThreshold w sesji MQSession, MQAX zgłasza wyjątek (numer 32000). Użyj tego w skrypcie, używając instrukcji On Error (lub równoważnej), aby ją przetworzyć.
3. Użyj funkcji Error, aby pobrać powiązany łańcuch błędu, który ma postać:

   ```
   MQAX: CompletionCode=xxx, ReasonCode=xxx, ReasonName=xxx
   ```

   Więcej informacji na temat używania instrukcji On Error można znaleźć w dokumentacji języka skryptowego ActiveX .

   Korzystanie z kodu CompletionCode i ReasonCode w obiekcie MQSession jest wygodne w przypadku prostych procedur obsługi błędów.

Właściwość ReasonName zwraca nazwę symboliczną IBM MQ dla bieżącej wartości parametru ReasonCode.

Zgłaszanie wyjątków

W poniższych regułach opisano sposób obsługi wyjątków:

- Za każdym razem, gdy właściwość lub metoda ustawia kod zakończenia na wartość większą lub równą progowi wyjątku (zwykle jest ustawiona na wartość 2), zgłaszany jest wyjątek.
- Wszystkie wywołania metod i zestawy właściwości ustawiają kod zakończenia.

Pobieranie właściwości

Jest to przypadek szczególny, ponieważ wartości CompletionCode i ReasonCode nie są zawsze aktualizowane:

- Jeśli właściwość zostanie pomyślnie zakończona, obiekt i obiekt MQSession ReasonCode i CompletionCode pozostaną niezmienione.
- Jeśli właściwość nie powiedzie się i zostanie wyświetlony kod CompletionCode ostrzeżenia, wartości ReasonCode i CompletionCode pozostaną niezmienione.
- Jeśli właściwość nie powiedzie się i zostanie zgłoszony błąd CompletionCode , wartości ReasonCode i CompletionCode zostaną zaktualizowane tak, aby odzwierciedlały prawdziwe wartości, a przetwarzanie błędów jest kontynuowane zgodnie z opisem.

Klasa MQSession ma metodę *ReasonCodeName* , która może zostać użyta do zastąpienia kodu przyczyny produktu IBM MQ nazwą symboliczną. Jest to szczególnie przydatne podczas tworzenia programów, w których mogą wystąpić nieoczekiwane błędy. Nazwa nie jest jednak idealna do prezentacji dla użytkowników.

Każda klasa ma również właściwość *ReasonName*, która zwraca nazwę symboliczną bieżącego kodu przyczyny dla tej klasy.

IBM MQ Klasy automatyzacji dla odwołania ActiveX

W tej sekcji opisano klasy klas automatyzacji programu IBM MQ dla ActiveX (MQAX), opracowane dla ActiveX. Klasy umożliwiają pisanie aplikacji ActiveX , które mogą uzyskiwać dostęp do innych aplikacji działających w środowiskach innych niżActiveX za pomocą programu IBM MQ.

Klasy automatyzacji produktu IBM MQ dla interfejsu ActiveX

Klasy automatyzacji produktu IBM MQ dla elementu ActiveX udostępniają predefiniowane stałe liczbowe ActiveX (takie jak MQMT_REQUEST), które są potrzebne do korzystania z klas.

Klasy automatyzacji ActiveX składają się z następujących elementów:

- “Klasa MQSession” na stronie 694
- “Klasa MQQueueManager” na stronie 697
- “Klasa MQQueue” na stronie 709
- “Klasa MQMessage” na stronie 724
- “Klasa opcji MQPutMessage” na stronie 747
- “Klasa opcji MQGetMessage” na stronie 749
- “Klasa MQDistributionList” na stronie 752
- “MQDistributionList-klasa elementu” na stronie 756

Ponadto klasy automatyzacji produktu IBM MQ dla elementu ActiveX udostępniają predefiniowane liczbowe stałe ActiveX (takie jak MQMT_REQUEST), które są potrzebne do korzystania z klas. Są one dostarczane w ramach typu wyliczeniowego MQ w bibliotece MQAX200. Stałe są podzbiorem

tych zdefiniowanych w plikach nagłówkowych języka C IBM MQ (cmqc * .h) z dodatkowymi klasami automatyzacji programu IBM MQ dla kodów przyczyny ActiveX .

Informacje o klasach automatyzacji produktu IBM MQ dla klas ActiveX

Informacje zawarte w tej sekcji znajdują się obok tematów referencyjnych w sekcji Tworzenie odwołań do aplikacji.

See Features that can be used only with the primary installation on Windows for important information.

Klasa MQSession udostępnia obiekt główny, który zawiera status ostatniego działania wykonywanego dla dowolnego z obiektów MQAX. Więcej informacji zawiera sekcja “Obsługa błędów” na stronie 691.

Klasy MQQueueManager i MQQueue zapewniają dostęp do bazowych obiektów produktu IBM MQ . Metody lub dostępy właściwości dla tych klas w ogólnym wyniku są wykonywane w ramach wywołań MQI produktu IBM MQ .

Klasy opcji MQMessage, MQPutMessagei MQGetMessageobudowują struktury danych MQMD, MQPMO i MQGMO i są używane w celu ułatwienia wysyłania komunikatów do kolejek i pobierania z nich komunikatów.

Klasa MQDistributionList hermetykuje kolekcję kolejek-lokalnych, zdalnych lub aliasów dla danych wyjściowych. Klasa elementu MQDistributionListhermetykuje struktury MQOR, MQRR i MQPMR i wiąże je z listą dystrybucyjną będącą właścicielem.

Przekazywanie parametrów

Parametry w wywołaniach metod są przekazywane przez wartość, z wyjątkiem sytuacji, w której parametr ten jest obiektem, w którym to przypadku jest to odniesienie, które jest przekazywane.

W definicjach klas podano listę typów danych dla każdego parametru lub właściwości. W przypadku wielu klientów ActiveX , takich jak Visual Basic, jeśli używana zmienna nie jest typu wymaganego, wartość jest automatycznie przekształcana w wymagany typ lub z wymaganego typu, a taka konwersja jest możliwa. Jest to zgodne ze standardowymi regułami klienta; MQAX nie zapewnia takiej konwersji.

Wiele metod jest typu łańcuchowego o stałej długości lub zwraca łańcuch znaków o stałej długości. Reguły konwersji są następujące:

- Jeśli użytkownik dostarcza łańcuch o stałej długości w niepoprawnej długości, jako parametr wejściowy lub jako wartość zwracaną, wartość jest obcinana lub dopełniona spacjami kończącymi się zgodnie z wymaganiami.
- Jeśli użytkownik dostarcza łańcuch o zmiennej długości o niepoprawnej długości jako parametr wejściowy, wartość jest obcinana lub dopełniona spacjami kończącymi.
- Jeśli użytkownik dostarcza łańcuch o zmiennej długości o niepoprawnej długości jako wartość zwracaną, to łańcuch jest dopasowywany do wymaganej długości (ponieważ zwracanie wartości niszczy poprzednią wartość w łańcuchu).
- Łańcuchy podane jako parametry wejściowe mogą zawierać osadzone Nulls.

Klasy te można znaleźć w bibliotece MQAX200 .

Metody dostępu do obiektów

Metody te nie są bezpośrednio związane z żadnym pojedynczym wywołaniem produktu IBM MQ . Każda z tych metod tworzy obiekt, w którym są przechowywane informacje o odwołaniu, a następnie następuje połączenie z obiektem IBM MQ lub jego otwarcie:

Po nawiązaniu połączenia z menedżerem kolejek atrybut uchwytu połączenia jest generowany przez produkt IBM MQ.

Po otwarciu kolejki atrybut "uchwyt obiektu" jest generowany przez program IBM MQ.

Te atrybuty IBM MQ nie są bezpośrednio dostępne dla programu MQAX.

Błędy

Błędy syntaktyczne podczas przekazywania parametrów mogą być wykrywane w czasie kompilacji i w czasie wykonywania przez klienta ActiveX . Błędy mogą zostać uwięzione przy użyciu opcji On Error w Visual Basic.

Wszystkie klasy programu IBM MQ ActiveX zawierają dwie specjalne właściwości tylko do odczytu- ReasonCode i CompletionCode. Właściwości te można odczytać w dowolnym momencie.

Próba uzyskania dostępu do dowolnej innej właściwości lub wywołanie dowolnego wywołania metody może spowodować wygenerowanie błędu z produktu IBM MQ.

Jeśli zestaw właściwości lub wywołanie metody zakończy się powodzeniem, parametr ReasonCode obiektu będącego właścicielem jest ustawiony na wartość MQRC_NONE, a CompletionCode jest ustawiony na wartość MQCC_OK.

Jeśli dostęp do właściwości lub wywołanie metody nie powiedzie się, w tych polach są ustawiane kody przyczyny i zakończenia.

Klasa MQSession

Jest to klasa główna dla klas automatyzacji produktu IBM MQ dla elementu ActiveX.

Dla każdego procesu klienta ActiveX zawsze istnieje tylko jeden obiekt MQSession. Próba utworzenia drugiego obiektu powoduje utworzenie drugiego odwołania do oryginalnego obiektu.

Tworzenie

Nowy -tworzy nowy obiekt MQSession.

Składnia

Dim *mqsess* **As New MQSession Set** *mqsess* = **Nowa sesja MQSession**

Właściwości

- “Właściwość CompletionCode” na stronie 694.
- “Właściwość ExceptionThreshold” na stronie 695.
- “Właściwość ReasonCode” na stronie 695.
- “Właściwość ReasonName” na stronie 696.

Metoda

- “Metoda AccessGetMessageOptions” na stronie 696.
- “Metoda AccessMessage” na stronie 696.
- “Metoda AccessPutMessageOptions” na stronie 696.
- “Metoda menedżera AccessQueue” na stronie 696.
- “ClearError-metoda kodów” na stronie 697.
- “Metoda ReasonCodeName” na stronie 697.

Właściwość CompletionCode

Tylko do odczytu. Zwraca kod zakończenia IBM MQ ustawiony za pomocą najnowszej metody lub zestawu właściwości wydanego dla dowolnego obiektu IBM MQ .

Jest resetowany do wywołania MQCC_OK, gdy metoda lub zestaw właściwości są pomyślnie wywoływane dla dowolnego obiektu MQAX.

Procedura obsługi zdarzeń błędów może sprawdzić tę właściwość w celu zdiagnozowania błędu, bez konieczności sprawdzania, który obiekt był zaangażowany.

Korzystanie z kodu CompletionCode i ReasonCode w obiekcie MQSession jest bardzo wygodne w przypadku prostych procedur obsługi błędów.

Uwaga: Sekcja "Wątki" na stronie 691 zawiera ograniczenia dotyczące używania kodów błędów MQSession w aplikacjach wielowątkowych.

Zdefiniowane w:
Klasa MQSession

Typ danych:
Long

Wartości:

- MQCC_OK
- MQCC_WARNING,
- MQCC_FAILED

Składnia:
Aby uzyskać informacje: *completioncode & = MQSession*.CompletionCode

Właściwość ExceptionThreshold

Odczyt-zapis. Definiuje poziom błędu IBM MQ , dla którego MQAX zgłosi wyjątek. Wartość domyślna to MQCC_FAILED. Wartość większa niż MQCC_FAILED skutecznie zapobiega przetwarzaniu wyjątków, pozostawiając programistę w celu wykonania sprawdzenia kodu CompletionCode i ReasonCode.

Zdefiniowane w: Klasa MQSession

Typ danych: Long

Wartości:

- Dowolna, ale należy wziąć pod uwagę wartość MQCC_WARNING, MQCC_FAILED lub większą.

Składnia:

Aby uzyskać: *ExceptionThreshold& = MQSession*. **ExceptionThreshold**

Aby ustawić wartość: *MQSession.* **ExceptionThreshold** = *ExceptionThreshold$*

Właściwość ReasonCode

Tylko do odczytu. Zwraca kod przyczyny ustawiony za pomocą najnowszej metody lub zestawu właściwości wydanego dla dowolnego obiektu IBM MQ .

Procedura obsługi zdarzeń błędów może sprawdzić tę właściwość w celu zdiagnozowania błędu, bez konieczności sprawdzania, który obiekt był zaangażowany.

Korzystanie z kodu CompletionCode i ReasonCode w obiekcie MQSession jest bardzo wygodne w przypadku prostych procedur obsługi błędów.

Uwaga: Sekcja "Wątki" na stronie 691 zawiera ograniczenia dotyczące używania kodów błędów MQSession w aplikacjach wielowątkowych.

Zdefiniowane w: Klasa MQSession

Typ danych: Long

Wartości:

- Patrz sekcja Przyczyna (MQLONG) oraz dodatkowe wartości MQAX wymienione w sekcji "Kody przyczyny IBM MQ Klasy automatyzacji dla ActiveX" na stronie 763.

Składnia: do pobrania: *kod_przyczyny & = Sesja MQSession*. **ReasonCode**

Właściwość ReasonName

Tylko do odczytu. Zwraca nazwę symboliczną najnowszego kodu przyczyny. Na przykład: "MQRC_QMGR_NOT_AVAILABLE".

Uwaga: Sekcja "Wątki" na stronie 691 zawiera ograniczenia dotyczące używania kodów błędów MQSession w aplikacjach wielowątkowych.

Zdefiniowane w: Klasa MQSession

Typ danych: String

Wartości:

- Patrz Kody zakończenia i przyczyny interfejsu API.

Składnia: aby uzyskać następujące informacje: *reasonname $= MQSession* **.ReasonName**

Metoda AccessGetMessageOptions

Tworzy nowy obiekt opcji MQGetMessage.

Zdefiniowane w:
Klasa MQSession

Składnia:
gmo = MQSession **.AccessGetMessageOptions()**

Metoda AccessMessage

Tworzy nowy obiekt MQMessage.

Zdefiniowane w:
Klasa MQSession

Składnia:
msg = MQSession **.AccessMessage()**

Metoda AccessPutMessageOptions

Tworzy nowy obiekt opcji MQPutMessage.

Zdefiniowane w:
Klasa MQSession

Składnia:
pmo = MQSession **.AccessPutMessageOptions()**

Metoda menedżera AccessQueue

Tworzy nowy obiekt MQQueueManager i łączy go z rzeczywistym menedżerem kolejek za pomocą serwera IBM MQ MQI client lub serwera IBM MQ . Podobnie jak w przypadku nawiązywania połączenia, ta metoda również wykonuje otwarte dla obiektu menedżera kolejek.

Jeśli zarówno serwer IBM MQ MQI client , jak i serwer IBM MQ są zainstalowane w systemie, aplikacje MQAX będą domyślnie uruchamiane dla serwera. Aby uruchomić program MQAX dla klienta, biblioteka powiązań klienta musi być określona w zmiennej środowiskowej `GMQ_MQ_LIB` , na przykład w polu `GMQ_MQ_LIB=mqic.dll`.

W przypadku instalacji tylko klienta nie jest konieczne ustawianie zmiennej środowiskowej `GMQ_MQ_LIB` . Jeśli ta zmienna nie zostanie ustawiona, program IBM MQ podejmie próbę załadowania pliku amqzst.dll. Jeśli ta biblioteka DLL nie jest obecna (jak ma to miejsce w przypadku instalacji tylko klienta), program IBM MQ podejmie próbę załadowania pliku mqic.dll.

If successful it sets the MQQueueManager's ConnectionStatus to TRUE.

Menedżer kolejek może być podłączony do co najwyżej jednego obiektu MQQueueManager na instancję ActiveX .

Jeśli nawiązanie połączenia z menedżerem kolejek nie powiedzie się, zostanie zgłoszone zdarzenie błędu, a obiekt MQSession ma ustawioną wartość ReasonCode i CompletionCode .

Zdefiniowane w: Klasa MQSession

Składnia: *set qm = MQSession* **.AccessQueueManager** (*Name$*)

Parametr: *Name$* Łańcuch. Nazwa menedżera kolejek, z którym ma zostać nawiązane połączenie.

ClearError-metoda kodów

Resetuje parametr CompletionCode do wartości MQCC_OK i ReasonCode na wartość MQRC_NONE.

Zdefiniowane w: Klasa MQSession

Składnia:

```
Call MQSession.ClearErrorCodes()
```

Metoda ReasonCodeName

Zwraca nazwę kodu przyczyny o podanej wartości liczbowej. Przydatne jest nadanie użytkownikom jaśniejszych wskazówek dotyczących warunków błędów. Nazwa jest nadal nieco kryptograficzna (na przykład ReasonCodeName (2059) to **MQRC_Q_MGR_NOT_AVAILABLE**), więc tam, gdzie to możliwe, należy wychwytywać błędy i zastąpić je tekstem opisowym odpowiednim dla aplikacji.

Zdefiniowane w: Klasa MQSession

Składnia: *errname $= MQSession* **.ReasonCodeNazwa** (*reasonCode&*)

Parametr: *kod_przyczyny &* Long. Kod przyczyny, dla którego wymagana jest nazwa symboliczna.

Klasa MQQueueManager

Ta klasa reprezentuje połączenie z menedżerem kolejek. Menedżer kolejek może być uruchomiony lokalnie (serwer IBM MQ) lub zdalnie z dostępem udostępnionym przez klienta IBM MQ . Aplikacja musi utworzyć obiekt tej klasy i połączyć ją z menedżerem kolejek. Gdy obiekt tej klasy zostanie zniszczony, zostanie automatycznie odłączony od jego menedżera kolejek.

Zawieranie

Obiekty klasy MQQueue są powiązane z tą klasą.

Nowy obiekt tworzy nowy obiekt MQQueueManager i ustawia wszystkie właściwości na wartości początkowe. Alternatywnie można użyć metody menedżera AccessQueueklasy MQSession.

Tworzenie

Nowy obiekt tworzy obiekt **nowy** MQQueueManager i ustawia wszystkie właściwości na wartości początkowe. Alternatywnie można użyć metody menedżera AccessQueueklasy MQSession.

Składnia

Dim *mgr* **As New MQQueueManager set** *mgr* = **New MQQueueManager**

Właściwości

- “Właściwość identyfikatora AlternateUser” na stronie 699.
- “Właściwość AuthorityEvent” na stronie 699.
- “Właściwość BeginOptions” na stronie 699.
- “Właściwość definicji ChannelAuto” na stronie 700.

Metody

Dostęp do właściwości

W dowolnym momencie można uzyskać dostęp do następujących właściwości.

- "Właściwość identyfikatora AlternateUser" na stronie 699.
- "Właściwość CompletionCode" na stronie 701.
- "Właściwość ConnectionStatus" na stronie 702.
- "Właściwość ReasonCode" na stronie 705.

Dostęp do pozostałych właściwości można uzyskać tylko wtedy, gdy obiekt jest połączony z menedżerem kolejek, a ID użytkownika jest uprawniony do sprawdzania się względem tego menedżera kolejek. Jeśli zostanie ustawiony alternatywny identyfikator użytkownika, a bieżący identyfikator użytkownika ma uprawnienia do jego używania, to zamiast tego zostanie sprawdzony alternatywny ID użytkownika w celu uzyskania autoryzacji.

Jeśli te warunki nie mają zastosowania, klasy automatyzacji programu IBM MQ dla elementu ActiveX podejmą próbę nawiązania połączenia z menedżerem kolejek i otworzenia go w celu automatycznego sprawdzenia. Jeśli ta operacja nie powiedzie się, wywołanie ustawia CompletionCode o wartości MQCC_FAILED i jeden z następujących elementów ReasonCodes:

- MQRC_CONNECTION_BROKEN
- MQRC_NOT_AUTHORIZED
- Błąd MQRC_Q_MGR_NAME_ERROR
- MQRC_Q_MGR_NOT_AVAILABLE

Właściwość identyfikatora AlternateUser

Odczyt-zapis. Alternatywny identyfikator użytkownika, który ma być używany do sprawdzania poprawności dostępu do atrybutów menedżera kolejek.

Ta właściwość nie może być ustawiona, jeśli właściwość IsConnected ma wartość TRUE.

Nie można ustawić tej właściwości w czasie, gdy obiekt jest otwarty.

Klasa **Defined in:** MQQueueManager

Data Type: Łańcuch o długości 12 znaków

Syntax: Aby uzyskać: *altuser $= MQQueueManager* **.AlternateUserId** , aby ustawić wartość: *MQQueueManager* **.AlternateUserId** = *altuser $*

Właściwość AuthorityEvent

Tylko do odczytu. Atrybut MQI AuthorityEvent .

Zdefiniowane w:
Klasa MQQueueManager

Typ danych:
Długa liczba całkowita

Wartości:

- MQEVR_DISABLED
- MQEVR_ENABLED

Składnia: Aby uzyskać informacje: *authevent = MQQueueManager* **.AuthorityEvent**

Właściwość BeginOptions

Odczyt-zapis. Są to opcje, które mają zastosowanie do metody Begin. Początkowo MQBO_NONE.

Zdefiniowane w:
Klasa MQQueueManager

Typ danych:
Długa liczba całkowita

Wartości:

- MQBO_NONE

Składnia: Aby uzyskać informacje: *beginoptions & =MQQueueManager*. **BeginOptions**

Aby ustawić: *MQQueueManager* **.BeginOptions** = *beginoptions &*

Właściwość definicji ChannelAuto

Tylko do odczytu. Określa, czy dozwolona jest automatyczna definicja kanału.

Zdefiniowane w:
Klasa MQQueueManager

Typ danych:
Długa liczba całkowita

Wartości:

- MQCHAD_DISABLED
- MQCHAD_ENABLED

Składnia: Aby uzyskać informacje: *channelautodef & = MQQueueManager*. **ChannelAutoDefinicja**

Właściwość ChannelAutoDefinitionEvent

Tylko do odczytu. Określa, czy generowane są zdarzenia automatycznej definicji kanału.

Zdefiniowane w:
Klasa MQQueueManager

Typ danych:
Długa liczba całkowita

Wartości:

- MQEVR_DISABLED
- MQEVR_ENABLED

Składnia: Aby uzyskać informacje: *channelautodefevent & =MQQueueManager*.
ChannelAutoDefinitionEvent

Właściwość ChannelAutoDefinitionExit

Tylko do odczytu. Nazwa wyjścia użytkownika używana dla definicji kanału automatycznego.

Zdefiniowane w:
Klasa MQQueueManager

Typ danych:
Łańcuch

Składnia: Aby uzyskać informacje: *channelautodefexit$= MQQueueManager*. **ChannelAutoDefinitionExit**

Właściwość CharacterSet

Tylko do odczytu. Atrybut MQI CodedCharSetId .

Zdefiniowane w: klasa MQQueueManager

Typ danych: Long

Składnia: do pobrania: *characterset & = MQQueueManager* **.CharacterSet**

Właściwość CloseOptions

Odczyt-zapis. Opcje używane do sterowania tym, co się dzieje, gdy menedżer kolejek jest zamknięty. Wartością początkową jest MQCO_NONE.

Zdefiniowane w:
Klasa MQQueueManager

Typ danych:
Długa liczba całkowita

Wartości:

- MQCO_NONE

Składnia: Aby uzyskać następujące informacje: *closeopt & = MQQueueManager* **.CloseOptions**

Aby ustawić następujące elementy: *MQQueueManager* **.CloseOptions** = *closeopt &*

Właściwość CommandInputQueueName

Tylko do odczytu. Atrybut QName CommandInputMQI.

Zdefiniowane w: klasa MQQueueManager

Typ danych: Łańcuch o długości 48 znaków

Składnia: aby uzyskać następujące informacje: *commandinputqname $= MQQueueManager* **.CommandInputQueueName**

Właściwość CommandLevel

Tylko do odczytu. Zwraca wersję i poziom implementacji menedżera kolejek produktu IBM MQ (atrybut MQI CommandLevel).

Zdefiniowane w: klasa MQQueueManager

Typ danych: Long

Składnia: Aby uzyskać: *level & = MQQueueManager* **.CommandLevel**

Właściwość CompletionCode

Tylko do odczytu. Zwraca kod zakończenia ustawiony przez ostatnią metodę lub dostęp do właściwości wystawiony dla obiektu.

Zdefiniowane w: klasa MQQueueManager

Typ danych: Long

Wartości:

- MQCC_OK
- MQCC_WARNING,
- MQCC_FAILED

Składnia: Aby uzyskać: *completioncode & = MQQueueManager* **.CompletionCode**

Właściwość ConnectionHandle

Tylko do odczytu. Uchwyt połączenia dla obiektu menedżera kolejek produktu IBM MQ .

Zdefiniowane w:
Klasa MQQueueManager

Typ danych:
Long

Składnia: Aby uzyskać: *hconn & = MQQueueManager.* **ConnectionHandle**

Właściwość ConnectionStatus

Tylko do odczytu. Wskazuje, czy obiekt jest połączony z menedżerem kolejek, czy nie.

Zdefiniowane w: klasa MQQueueManager

Typ danych: Wartość boolowska

Wartości:

- PRAWDA (-1)
- FALSE (0)

Składnia: Aby uzyskać: *status = MQQueueManager* **.ConnectionStatus**

Właściwość ConnectOptions

Odczyt-Zapis. Te opcje mają zastosowanie do metody Connect. Początkowo MQCNO_NONE.

Zdefiniowane w:
Klasa MQQueueManager

Typ danych:
Długa liczba całkowita

Wartości:

- MQCNO_STANDARD_BINDING
- MQCNO_FASTPATH_BINDING
- MQCNO_NONE

Składnia: do pobrania: *connectoptions & =MQQueueManager*. **ConnectOptions**

Aby ustawić następujące elementy: *MQQueueManager* **.ConnectOptions** = *connectoptions &*

Właściwość DeadLetterQueueName

Tylko do odczytu. Atrybut nazwy QName DeadLetterMQI.

Zdefiniowane w: klasa MQQueueManager

Typ danych: Łańcuch o długości 48 znaków

Składnia: do pobrania: *dlqname $= MQQueueManager* **.DeadLetterQueueName**

Właściwość DefaultTransmissionQueueName

Tylko do odczytu. Atrybut nazwy QName DefXmitMQI.

Zdefiniowane w: klasa MQQueueManager

Typ danych: Łańcuch o długości 48 znaków

Składnia: Aby uzyskać: *defxmitqname $= MQQueueManager* **.DefaultTransmissionQueueName**

Opis, właściwość

Tylko do odczytu. Atrybut QMgrDesc MQI.

Zdefiniowane w: klasa MQQueueManager

Typ danych: Łańcuch o długości 64 znaków

Składnia: Aby uzyskać: *description $= MQQueueManager* **.Opis**

Właściwość DistributionLists

Tylko do odczytu. Jest to zdolność menedżera kolejek do obsługi list dystrybucyjnych.

Zdefiniowane w:
Klasa MQQueueManager

Typ danych:
wartość boolowska

Wartości:

- PRAWDA (-1)
- FALSE (0)

Składnia: Aby uzyskać informacje: *distributionlists= MQQueueManager*. **DistributionLists**

Właściwość InhibitEvent

Tylko do odczytu. Atrybut MQI InhibitEvent .

Zdefiniowane w: klasa MQQueueManager

Typ danych: Long

Wartości:

- MQEVR_DISABLED
- MQEVR_ENABLED

Składnia: Aby uzyskać informacje: *inhibevent & = MQQueueManager* **.InhibitEvent**

Właściwość IsConnected

Wartość wskazująca, czy menedżer kolejek jest aktualnie połączony.

Tylko do odczytu.

Zdefiniowane w: klasa MQQueueManager

Typ danych: Wartość boolowska

Wartości:

- PRAWDA (-1)
- FALSE (0)

Składnia: Aby uzyskać: *isconnected = MQQueueManager* **.IsConnected**

Właściwość IsOpen

Wartość wskazująca, czy menedżer kolejek jest aktualnie otwarty na potrzeby zapytania.

Tylko do odczytu.

Zdefiniowane w:
Klasa MQQueueManager

Typ danych:
wartość boolowska

Wartości:

- PRAWDA (-1)
- FALSE (0)

Składnia: Aby uzyskać: *IsOpen = MQQueueManager*. **IsOpen**

Właściwość LocalEvent

Tylko do odczytu. Atrybut MQI LocalEvent .

Zdefiniowane w: klasa MQQueueManager

Typ danych: Long

Wartości:

- MQEVR_DISABLED
- MQEVR_ENABLED

Składnia: Aby uzyskać informacje: *localevent & = MQQueueManager* **.LocalEvent**

Właściwość MaximumHandles

Tylko do odczytu. Atrybut MaxHandles MQI.

Zdefiniowane w: klasa MQQueueManager

Typ danych: Long

Składnia: Aby uzyskać: *maxUchwyty & = MQQueueManager* **.MaximumHandles**

Właściwość Length MaximumMessage

Tylko do odczytu. Atrybut menedżera kolejek długości MaxMsgMQI.

Zdefiniowane w: klasa MQQueueManager

Typ danych: Long

Składnia: Aby uzyskać informacje: *maxmessagelength & = MQQueueManager* **.MaximumMessageLength**

Właściwość MaximumPriority

Tylko do odczytu. Atrybut MaxPriority MQI.

Zdefiniowane w: klasa MQQueueManager

Typ danych: Long

Składnia: Aby uzyskać: *maxpriority & = MQQueueManager* **.MaximumPriority**

Właściwość Komunikaty MaximumUncommitted

Tylko do odczytu. Atrybut MaxUncommittedkomunikatów MQI.

Zdefiniowane w: klasa MQQueueManager

Typ danych: Long

Składnia: Aby uzyskać następujące informacje: *maxuncommitted & = MQQueueManager* **.MaximumUncommittedKomunikaty**

Właściwość Nazwa

Odczyt-zapis. Atrybut MQI QMgrName . Tej właściwości nie można zapisać po nawiązaniu połączenia z programem MQQueueManager .

Zdefiniowane w: klasa MQQueueManager

Typ danych: Łańcuch o długości 48 znaków

Składnia: aby uzyskać: *name $= MQQueueManager* **.name**

Aby ustawić: *MQQueueManager* **.name** = *nazwa $*

Uwaga: Visual Basic rezerwuje właściwość "Name" na potrzeby użycia w interfejsie wizualnym. Dlatego w przypadku używania w języku Visual Basic użyj dolnego przypadku, tj. "name".

Właściwość ObjectHandle

Tylko do odczytu. Uchwyt obiektu dla obiektu menedżera kolejek produktu IBM MQ .

Zdefiniowane w:
Klasa MQQueueManager

Typ danych
Long

Składnia: Aby uzyskać: *hobj & = MQQueueManager.* **ObjectHandle**

Właściwość PerformanceEvent

Tylko do odczytu. Atrybut MQI PerformanceEvent .

Zdefiniowane w: klasa MQQueueManager

Typ danych: Long

Wartości:

- MQEVR_DISABLED
- MQEVR_ENABLED

Składnia: Aby uzyskać następujące informacje: *perfevent & = MQQueueManager*.PerformanceEvent

Właściwość platformy

Tylko do odczytu. Atrybut platformy MQI.

Zdefiniowane w: klasa MQQueueManager

Typ danych: Long

Wartości:

- MQPL_WINDOWS_NT
- MQPL_WINDOWS

Składnia: Aby uzyskać: *platforma & = MQQueueManager* **.Platforma**

Właściwość ReasonCode

Tylko do odczytu. Zwraca kod przyczyny ustawiony przez ostatnią metodę lub dostęp do właściwości wystawiony dla obiektu.

Zdefiniowane w: klasa MQQueueManager

Typ danych: Long

Wartości:

- Patrz Kody zakończenia i przyczyny interfejsu API.

Składnia: do pobrania: *kod_przyczyny & = MQQueueManager* **.ReasonCode**

Właściwość ReasonName

Tylko do odczytu. Zwraca nazwę symboliczną najnowszego kodu przyczyny. Na przykład: "MQRC_QMGR_NOT_AVAILABLE".

Zdefiniowane w: klasa MQQueueManager

Typ danych: String

Wartości:

- Patrz Kody zakończenia i przyczyny interfejsu API.

Składnia: aby uzyskać następujące informacje: *reasonname $= MQQueueManager* **.ReasonName**

Właściwość RemoteEvent

Tylko do odczytu. Atrybut MQI RemoteEvent .

Zdefiniowane w: klasa MQQueueManager

Typ danych: Long

Wartości:

- MQEVR_DISABLED
- MQEVR_ENABLED

Składnia: Aby uzyskać następujące informacje: *remoteevent & = MQQueueManager* **.RemoteEvent**

Właściwość zdarzenia StartStop

Tylko do odczytu. Atrybut zdarzenia StartStopMQI.

Zdefiniowane w: klasa MQQueueManager

Typ danych: Long

Wartości:

- MQEVR_DISABLED
- MQEVR_ENABLED

Składnia: Aby uzyskać informacje: *strstpevent & = MQQueueManager* **.StartStopEvent**

Właściwość Dostępność SyncPoint

Tylko do odczytu. Atrybut SyncPoint MQI.

Zdefiniowane w: klasa MQQueueManager

Typ danych: Long

Wartości:

- MQSP_AVAILABLE
- MQSP_NOT_AVAILABLE

Składnia: Aby uzyskać informacje: *syncpointavailability & = MQQueueManager* **.SyncPointDostępność**

Właściwość TriggerInterval

Tylko do odczytu. Atrybut MQI TriggerInterval .

Zdefiniowane w: klasa MQQueueManager

Typ danych: Long

Składnia: Aby uzyskać następujące informacje: *trigint & = MQQueueManager* **.TriggerInterval**

Metoda AccessQueue

Tworzy obiekt MQQueue i wiąże go z tym obiektem MQQueueManager , ustawiając właściwość odwołania połączenia dla kolejki. Ustawia on właściwości Nazwa, OpenOptions, DynamicQueueName i AlternateUserobiektu MQQueue na podane wartości i próbuje je otworzyć.

Jeśli otwarcie nie powiedzie się, wywołanie nie powiedzie się. Zdarzenie błędu jest zgłaszane dla obiektu. Ustawione są wartości ReasonCode i CompletionCodeoraz MQSession ReasonCode i CompletionCode obiektu.

Parametry DynamicQueueName, QueueManagerName i AlternateUserId są opcjonalne i domyślne dla wartości "".

Należy określić opcję OpenOption MQOO_INQUIRE oprócz innych opcji, jeśli właściwości kolejki mają zostać odczytane.

Nie należy ustawiać nazwy QueueManagerani ustawić jej na wartość "", jeśli kolejka, która ma zostać otwarta, jest lokalna. W przeciwnym razie należy ustawić nazwę menedżera kolejek zdalnych, który jest właścicielem kolejki, a następnie podjąć próbę otwarcia lokalnej definicji kolejki zdalnej. Więcej informacji na temat rozstrzygania nazw kolejek zdalnych i aliasów menedżera kolejek zawiera sekcja Jakie są aliasy?.

Jeśli właściwość Nazwa jest ustawiona na nazwę kolejki modelowej, należy określić nazwę kolejki dynamicznej, która ma zostać utworzona w parametrze Name$ DynamicQueue$. Jeśli wartość podana w parametrze Name$ DynamicQueuema wartość "", to wartość ustawiona w obiekcie kolejki i używana w wywołaniu open to "AMQ. *". Więcej informacji na temat nazw kolejek dynamicznych zawiera sekcja “Tworzenie kolejek dynamicznych” na stronie 842 .

Definicja

Zdefiniowane w: klasa MQQueueManager .

Składnia

Składnia: ustaw kolejkę = MQQueueManager. **AccessQueue** (Name$, OpenOptions&, QueueManagerName$, DynamicQueueName$, AlternateUserId$)

Parametry

Nazwa$ Łańcuch. Nazwa kolejki produktu IBM MQ .

OpenOptions: Long. Opcje, które mają być używane, gdy kolejka jest otwarta. Patrz OpenOptions (MQLONG).

QueueManagerNazwa $ Łańcuch. Nazwa menedżera kolejek, który jest właścicielem kolejki, która ma zostać otwarta. Wartość "" oznacza, że menedżer kolejek jest lokalny.

*DynamicQueueNazwa$* String. Nazwa przypisana do kolejki dynamicznej w momencie otwarcia kolejki, gdy parametr Name$ określa kolejkę modelową.

AlternateUserId$ Łańcuch. Alternatywny identyfikator użytkownika używany do sprawdzania poprawności dostępu podczas otwierania kolejki.

Metoda listy AddDistribution

Tworzy nowy obiekt MQDistributionList i ustawia jego odwołanie do połączenia z menedżerem kolejek będącym właścicielem.

Zdefiniowane w:
Klasa MQQueueManager

Składnia: *set distributionlist* = **MQQueueManager.** *ListaAddDistribution*

Metoda wycofania

Tworzy kopię zapasową wszystkich niezatwierdzonych operacji umieszczania i pobierania komunikatów, które wystąpiły jako część jednostki pracy od ostatniego punktu synchronizacji.

Zdefiniowane w: klasa MQQueueManager

Składnia:

```
Call MQQueueManager.Backout()
```

Metoda rozpoczęcia

Rozpoczyna jednostkę pracy, która jest koordynowana przez menedżer kolejek. Opcje początku mają wpływ na zachowanie tej metody.

Zdefiniowane w:
Klasa MQQueueManager

Składnia:

```
Call MQQueueManager.Begin()
```

ClearError-metoda kodów

Resetuje parametr CompletionCode do wartości MQCC_OK i ReasonCode na wartość MQRC_NONE zarówno dla klasy MQQueueManager , jak i klasy MQSession.

Zdefiniowane w:
Klasa MQQueueManager

Składnia:
Call *MQQueueManager* **.ClearErrorCodes ()**

Metoda zatwierdzania

Zatwierdza wszelkie operacje umieszczania i pobierania komunikatów, które wystąpiły jako część jednostki pracy od ostatniego punktu synchronizacji.

Zdefiniowane w: klasa MQQueueManager

Składnia:

```
Call MQQueueManager.Commit()
```

Metoda połączenia

Łączy obiekt MQQueueManager z rzeczywistym menedżerem kolejek za pośrednictwem serwera IBM MQ MQI client lub serwera. Podobnie jak w przypadku nawiązywania połączenia, ta metoda również otwiera obiekt menedżera kolejek, aby można było go odpytywać.

Ustawia wartość parametru IsConnected na wartość TRUE.

Maksymalnie jeden obiekt MQQueueManager dla instancji ActiveX może łączyć się z menedżerem kolejek.

Zdefiniowane w: klasa MQQueueManager

Składnia:

```
Call MQQueueManager.Connect()
```

Metoda rozłączania

Odłącza obiekt MQQueueManager od menedżera kolejek.

Ustawia wartość IsConnected na FALSE.

Wszystkie obiekty kolejki powiązane z obiektem MQQueueManager nie mogą być używane i nie można ich ponownie otworzyć.

Wszystkie niezatwierdzone zmiany (operacje umieszczania i pobierania komunikatów) są zatwierdzane.

Zdefiniowane w: klasa MQQueueManager

Składnia:

```
Call MQQueueManager.Disconnect()
```

Klasa MQQueue

Ta klasa reprezentuje dostęp do kolejki produktu IBM MQ . To połączenie jest udostępniane przez powiązany obiekt MQQueueManager . Gdy obiekt tej klasy zostanie zniszczony, zostanie automatycznie zamknięty.

Zawieranie

Klasa MQQueue jest zawarta w klasie MQQueueManager .

Tworzenie

Produkt New tworzy nowy obiekt MQQueue i ustawia wszystkie właściwości na wartości początkowe. Alternatywnie można użyć metody AccessQueue klasy MQQueueManager .

Składnia

```
Dim que As New MQQueue Set que = New MQQueue
```

Właściwości

- “Właściwość identyfikatora AlternateUser” na stronie 711.
- “Właściwość Nazwa BackoutRequeue” na stronie 712.
- “Właściwość BackoutThreshold” na stronie 712.
- “Właściwość Nazwa BaseQueue” na stronie 712.
- “Właściwość CloseOptions” na stronie 712.
- “Właściwość CompletionCode” na stronie 712.
- “Właściwość ConnectionReference” na stronie 713.
- “Właściwość czasu CreationDate” na stronie 713.
- “Właściwość CurrentDepth” na stronie 713.
- “Właściwość DefaultInputOpenOption” na stronie 713.
- “Właściwość DefaultPersistence” na stronie 713.
- “Właściwość DefaultPriority” na stronie 713.
- “Właściwość DefinitionType” na stronie 714.
- “Właściwość zdarzenia DepthHigh” na stronie 714.
- “Właściwość DepthHighLimit” na stronie 714.
- “Właściwość zdarzenia DepthLow” na stronie 714.
- “DepthLow-właściwość Limit” na stronie 714.
- “Właściwość zdarzenia DepthMaximum” na stronie 715.
- “Właściwość zdarzenia DepthHigh” na stronie 714.
- “Właściwość DepthHighLimit” na stronie 714.
- “Właściwość zdarzenia DepthLow” na stronie 714.
- “DepthLow-właściwość Limit” na stronie 714.
- “Właściwość zdarzenia DepthMaximum” na stronie 715.
- “Opis, właściwość” na stronie 715.
- “Właściwość Nazwa DynamicQueue” na stronie 715.
- “HardenGet, właściwość Backout” na stronie 715.
- “Właściwość InhibitGet” na stronie 715.

- “Właściwość InhibitPut” na stronie 716.
- “Właściwość Nazwa InitiationQueue” na stronie 716.
- “Właściwość IsOpen” na stronie 716.
- “Właściwość MaximumDepth” na stronie 716.
- “Właściwość Length MaximumMessage” na stronie 716.
- “Właściwość sekwencji MessageDelivery” na stronie 717.
- “Właściwość ObjectHandle” na stronie 717.
- “Właściwość Liczba OpenInput” na stronie 717.
- “Właściwość OpenOptions” na stronie 717.
- “Właściwość Liczba OpenOutput” na stronie 718.
- “Właściwość OpenStatus” na stronie 718.
- “Właściwość ProcessName” na stronie 718.
- “Właściwość Nazwa QueueManager” na stronie 718.
- “Właściwość QueueType” na stronie 718.
- “Właściwość ReasonCode” na stronie 719.
- “Właściwość ReasonName” na stronie 719.
- “Właściwość RemoteQueueManagerName” na stronie 719.
- “Właściwość Nazwa RemoteQueueName” na stronie 719.
- “Właściwość ResolvedQueueManagerName” na stronie 719.
- “Właściwość Nazwa ResolvedQueue” na stronie 719.
- “Właściwość RetentionInterval” na stronie 720.
- “Właściwość zasięgu” na stronie 720.
- “Właściwość ServiceInterval” na stronie 720.
- “Właściwość zdarzenia ServiceIntervalEvent” na stronie 720.
- “Właściwość współużytkowalności” na stronie 720.
- “Właściwość Nazwa kolejki TransmissionQueue” na stronie 721.
- “Właściwość TriggerControl” na stronie 721.
- “Właściwość TriggerData” na stronie 721.
- “Właściwość TriggerDepth” na stronie 721.
- “Właściwość priorytetu TriggerMessage” na stronie 721.
- “Właściwość TriggerType” na stronie 721.
- “właściwość użycia” na stronie 722.

Metody

- “ClearError-metoda kodów” na stronie 722
- “Metoda zamknięcia” na stronie 722
- “metoda GET” na stronie 722
- “Metoda otwierania” na stronie 723
- “metoda PUT” na stronie 723

Dostęp do właściwości

Jeśli obiekt kolejki nie jest połączony z menedżerem kolejek, można zapoznać się z następującymi właściwościami:

- “Właściwość CompletionCode” na stronie 712
- “Właściwość OpenStatus” na stronie 718
- “Właściwość ReasonCode” na stronie 719

i można odczytywać i zapisywać do:

- “Właściwość identyfikatora AlternateUser” na stronie 711
- “Właściwość CloseOptions” na stronie 712
- “Właściwość ConnectionReference” na stronie 713
- “Właściwość Nazwa” na stronie 717
- “Właściwość OpenOptions” na stronie 717

Jeśli obiekt kolejki jest połączony z menedżerem kolejek, można odczytać wszystkie właściwości.

Właściwości atrybutu kolejki

Właściwości, które nie zostały wymienione w poprzedniej sekcji, to wszystkie atrybuty bazowej kolejki produktu IBM MQ . Dostęp do nich można uzyskać tylko wtedy, gdy obiekt jest połączony z menedżerem kolejek, a ID użytkownika jest autoryzowany do uzyskiwania informacji lub ustawiania dla tej kolejki. Jeśli zostanie ustawiony alternatywny identyfikator użytkownika, a bieżący identyfikator użytkownika ma uprawnienia do jego używania, to zamiast niego zostanie sprawdzony alternatywny ID użytkownika.

Właściwość musi być odpowiednią właściwością dla danego typu QueueType. Więcej informacji na ten temat zawiera sekcja Atrybuty kolejek .

Jeśli te warunki nie mają zastosowania, dostęp do właściwości ustawi obiekt CompletionCode o wartości MQCC_FAILED i jeden z następujących elementów ReasonCodes:

- MQRC_CONNECTION_BROKEN
- MQRC_NOT_AUTHORIZED
- Błąd MQRC_Q_MGR_NAME_ERROR
- MQRC_Q_MGR_NOT_CONNECTED
- MQRC_SELECTOR_NOT_FOR_TYPE (CompletionCode to MQCC_WARNING)

Otwieranie kolejki

Jedynym sposobem utworzenia obiektu MQQueue jest użycie metody MQQueueManager AccessQueue lub nowego obiektu. Otwarty obiekt MQQueue pozostaje otwarty (OpenStatus= TRUE), dopóki nie zostanie zamknięty lub usunięty lub dopóki obiekt menedżera kolejek nie zostanie usunięty lub połączenie zostanie utracone do menedżera kolejek. Wartość właściwości CloseOptions kolejki MQQueue steruje zachowaniem operacji zamykania, która ma miejsce, gdy obiekt MQQueue zostanie usunięty.

Metoda MQQueueManager AccessQueue otwiera kolejkę za pomocą parametru OpenOptions . Metoda MQQueue.Open otwiera kolejkę przy użyciu właściwości OpenOptions . Program IBM MQ sprawdza poprawność opcji OpenOptions względem autoryzacji użytkownika w ramach procesu kolejki otwartej.

Właściwość identyfikatora AlternateUser

Odczyt-zapis. Alternatywny identyfikator użytkownika używany do sprawdzania poprawności dostępu do kolejki po jego otwarciu.

Nie można ustawić tej właściwości w czasie, gdy obiekt jest otwarty (to znaczy, gdy parametr IsOpen ma wartość TRUE).

Zdefiniowane w: Klasa MQQueue

Typ danych: Łańcuch o długości 12 znaków

Składnia: Aby uzyskać: *altuser $= MQQueue* **.AlternateUserId**

Ustawienie: *MQQueue.* **AlternateUserId** = *altuser $*

Właściwość Nazwa BackoutRequeue

Tylko do odczytu. Atrybut BackOutRequeueQName MQI.

Zdefiniowane w: Klasa MQQueue

Typ danych: Łańcuch o długości 48 znaków

Składnia: Aby uzyskać: *backoutrequeuename $= MQQueue* **.BackoutRequeueNazwa**

Właściwość BackoutThreshold

Tylko do odczytu. Atrybut BackoutThreshold MQI.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Wartości:

- Patrz BackoutThreshold (MQLONG).

Składnia: Aby uzyskać: *backoutthreshold & = MQQueue.* **BackoutThreshold**

Właściwość Nazwa BaseQueue

Tylko do odczytu. Nazwa kolejki, do której alias jest tłumaczona.

Poprawna tylko dla kolejek aliasowych.

Zdefiniowane w: Klasa MQQueue

Typ danych: Łańcuch o długości 48 znaków

Składnia: aby uzyskać: *baseqname $= MQQueue* **.BaseQueueName**

Właściwość CloseOptions

Odczyt-Zapis. Opcje używane do sterowania tym, co się dzieje, gdy kolejka jest zamknięta.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Wartości:

- MQCO_NONE
- MQCO_DELETE
- MQCO_DELETE_PURGE

Komendy MQCO_DELETE i MQCO_DELETE_PURGE są poprawne tylko dla kolejek dynamicznych.

Składnia: Aby uzyskać następujące informacje: *closeopt & = MQQueue* **.CloseOptions**

Aby ustawić: *MQQueue* **.CloseOptions** = *closeopt &*

Właściwość CompletionCode

Tylko do odczytu. Zwraca kod zakończenia ustawiony przez ostatnią metodę lub dostęp do właściwości wystawiony dla obiektu.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Wartości:

- MQCC_OK
- MQCC_WARNING,
- MQCC_FAILED

Składnia: Aby uzyskać: *completioncode & = MQQueue* **.CompletionCode**

Właściwość ConnectionReference

Odczyt-zapis. Definiuje obiekt menedżera kolejek, do którego należy obiekt kolejki. Nie można zapisać odwołania do połączenia w czasie, gdy kolejka jest otwarta.

Zdefiniowane w: Klasa MQQueue

Typ danych: MQQueueManager

Wartości:

- Odwołanie do aktywnego obiektu menedżera kolejek produktu IBM MQ

Składnia: aby ustawić: *set MQQueue* **.ConnectionReference** = *ConnectionReference*

Aby uzyskać następujące informacje: *set ConnectionReference = MQQueue* **.ConnectionReference**

Właściwość czasu CreationDate

Tylko do odczytu. Data i godzina utworzenia tej kolejki.

Zdefiniowane w: Klasa MQQueue

Typ danych: Variant typu 7 (data/godzina) lub EMPTY

Składnia: Aby uzyskać: *datetime = MQQueue* **.CreationDate**

Właściwość CurrentDepth

Tylko do odczytu. Liczba komunikatów znajdujących się obecnie w kolejce.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Składnia: aby uzyskać następujące informacje: *currentdepth & = MQQueue* **.CurrentDepth**

Właściwość DefaultInputOpenOption

Tylko do odczytu. Określa sposób otwierania kolejki, jeśli opcja OpenOptions określa wartość MQOO_INPUT_AS_Q_DEF.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Wartości:

- MQOO_INPUT_EXCLUSIVE
- MQOO_INPUT_SHARED

Składnia: Aby uzyskać następujące informacje: *defaultinop & = MQQueue* **.DefaultInputOpenOption**

Właściwość DefaultPersistence

Tylko do odczytu. Domyślna trwałość komunikatów w kolejce.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Składnia: Aby uzyskać następujące informacje: *defpersistence & = MQQueue* **.DefaultPersistence**

Właściwość DefaultPriority

Tylko do odczytu. Domyślny priorytet komunikatów w kolejce.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Składnia: Aby uzyskać następujący element: *defpriority & = MQQueue* **.DefaultPriority**

Właściwość DefinitionType

Tylko do odczytu. Typ definicji kolejki.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Wartości:

- MQQDT_PREDEFINIOWANE
- MQQDT_PERMANENT_DYNAMIC
- MQQDT_TEMPORARY_DYNAMIC

Składnia: do pobrania: *deftype & = MQQueue* **.DefinitionType**

Właściwość zdarzenia DepthHigh

Tylko do odczytu. Atrybut zdarzenia QDepthHighMQI.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Wartości:

- MQEVR_DISABLED
- MQEVR_ENABLED

Składnia: Aby uzyskać: *depthhighevent & = MQQueue.* **DepthHighZdarzenie**

Właściwość DepthHighLimit

Tylko do odczytu. Atrybut Limit MQI QDepthHigh.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Składnia: Aby uzyskać: *depthhighlimit & = MQQueue.* **DepthHighLimit**

Właściwość zdarzenia DepthLow

Tylko do odczytu. Atrybut zdarzenia QDepthLowMQI.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Wartości:

- MQEVR_DISABLED
- MQEVR_ENABLED

Składnia: Aby uzyskać: *depthlowevent & = MQQueue.* **DepthLowZdarzenie**

DepthLow-właściwość Limit

Tylko do odczytu. Atrybut Limit MQI QDepthLow.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Składnia: Aby uzyskać: *depthlowlimit & = MQQueue.* **DepthLowLimit**

Właściwość zdarzenia DepthMaximum

Tylko do odczytu. Atrybut zdarzenia QDepthMaxMQI.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Wartości:

- MQEVR_DISABLED
- MQEVR_ENABLED

Składnia: Aby uzyskać: *depthmaximumevent & = MQQueue.* **DepthMaximumZdarzenie**

Opis, właściwość

Tylko do odczytu. Opis kolejki.

Zdefiniowane w: Klasa MQQueue

Typ danych: Łańcuch o długości 64 znaków

Składnia: Aby uzyskać następujący tekst: *description $= MQQueue* **.Opis**

Właściwość Nazwa DynamicQueue

Odczyt-zapis, tylko do odczytu, gdy kolejka jest otwarta.

Ta opcja steruje nazwą kolejki dynamicznej używanej podczas otwierania kolejki modelowej. Można go ustawić za pomocą znaku wieloznacznego przez użytkownika jako zestaw właściwości (tylko wtedy, gdy kolejka jest zamknięta) lub jako parametr w parametrze MQQueueManager.AccessQueue().

Rzeczywista nazwa kolejki dynamicznej znajduje się w zapytaniu QueueName.

Zdefiniowane w: Klasa MQQueue

Typ danych: Łańcuch o długości 48 znaków

Wartości:

- Dowolna poprawna nazwa kolejki produktu IBM MQ .

Składnia: do ustawienia: *MQQueue* **.DynamicQueueName** = *dynamicqueuename $*

Aby uzyskać następujące informacje: *dynamicqueuename $ = MQQueue* **.DynamicQueueName**

HardenGet, właściwość Backout

Tylko do odczytu. Określa, czy zachować dokładną liczbę wycofanych danych.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Wartości:

- MQQA_BACKOUT_HARTOWANA
- MQQA_BACKOUT_NOT HARTOWANE

Składnia: Aby uzyskać informacje: *hardengetback & = MQQueue* **.HardenGetBackout**

Właściwość InhibitGet

Odczyt-zapis. Atrybut InhibitGet MQI.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Wartości:

- MQQA_GET_INHIBITED
- MQQA_GET_ALLOWED

Składnia: Aby uzyskać: *getstatus & = MQQueue* **.InhibitGet**

Aby ustawić: *MQQueue* **.InhibitGet** = *getstatus &*

Właściwość InhibitPut

Odczyt-zapis. Atrybut MQI InhibitPut .

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Wartości:

- MQQA_PUT_INHIBITED
- MQQA_PUT_ALLOWED

Składnia: Aby uzyskać: *putstatus & = MQQueue* **.InhibitPut**

Aby ustawić: *MQQueue* **.InhibitPut** = *putstatus &*

Właściwość Nazwa InitiationQueue

Tylko do odczytu. Nazwa kolejki inicjuj.

Zdefiniowane w: Klasa MQQueue

Typ danych: Łańcuch o długości 48 znaków

Składnia: Aby uzyskać: *initqname $= MQQueue* **.InitiationQueueNazwa**

Właściwość IsOpen

Zwraca informację o tym, czy kolejka jest otwarta.

Tylko do odczytu.

Zdefiniowane w: Klasa MQQueue

Typ danych: Wartość boolowska

Wartości:

- PRAWDA (-1)
- FALSE (0)

Składnia: Aby uzyskać: *open = MQQueue* **.IsOpen**

Właściwość MaximumDepth

Tylko do odczytu. Maksymalna głębokość kolejki.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Składnia: Aby uzyskać następujące informacje: *maxdepth & = MQQueue* **.MaximumDepth**

Właściwość Length MaximumMessage

Tylko do odczytu. Maksymalna dozwolona długość komunikatu w bajtach dla tej kolejki.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Składnia: Aby uzyskać informacje: *maxmlength & = MQQueue* **.MaximumMessageLength**

Właściwość sekwencji MessageDelivery

Tylko do odczytu. Kolejność dostarczania komunikatów.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Wartości:

- MQMDS_PRIORITY,
- MQMDS_FIFO

Składnia: Aby uzyskać: *messdelseq & = MQQueue* **.MessageDelivery, kolejność**

Właściwość Nazwa

Odczyt-zapis. Atrybut kolejki MQI. Ta właściwość nie może zostać zapisana po otwarciu kolejki MQQueue.

Zdefiniowane w: Klasa MQQueue

Typ danych: Łańcuch o długości 48 znaków

Składnia: aby uzyskać: *name $= MQQueue* **.name**

Aby ustawić: *MQQueue* **.name** = *nazwa $*

Uwaga: Visual Basic rezerwuje właściwość "Name" na potrzeby użycia w interfejsie wizualnym. Dlatego w przypadku używania w języku Visual Basic użyj dolnego przypadku, tj. "name".

Właściwość ObjectHandle

Tylko do odczytu. Uchwyt obiektu dla obiektu kolejki IBM MQ .

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Składnia: Aby uzyskać: *hobj & = MQQueue.* **ObjectHandle**

Właściwość Liczba OpenInput

Tylko do odczytu. Liczba operacji otwierania dla danych wejściowych.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Składnia: Aby uzyskać:

```
openincount& = MQQueue.OpenInputCount
```

Właściwość OpenOptions

Odczyt-zapis. Opcje, które mają być używane do otwierania kolejki.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Wartości:

- Patrz OpenOptions (MQLONG).

Składnia: Aby uzyskać:

```
openopt& = MQQueue.OpenOptions
```

Ustawienie: *MQQueue.* **OpenOptions** = *openopt &*

Właściwość Liczba OpenOutput

Tylko do odczytu. Liczba operacji otwierania dla danych wyjściowych.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Składnia: Aby uzyskać:

```
openoutcount& = MQQueue.OpenOutputCount
```

Właściwość OpenStatus

Tylko do odczytu. Wskazuje, czy kolejka jest otwarta, czy nie. Wartością początkową jest TRUE po metodzie AccessQueue lub wartość FALSE po Nowym.

Zdefiniowane w: Klasa MQQueue

Typ danych: Wartość boolowska

Wartości:

- PRAWDA (-1)
- FALSE (0)

Składnia: Aby uzyskać:

```
status& = MQQueue.OpenStatus
```

Właściwość ProcessName

Tylko do odczytu. Atrybut ProcessName MQI.

Zdefiniowane w: Klasa MQQueue

Typ danych: Łańcuch o długości 48 znaków

Składnia: Do pobrania: *nazwa_proc_$ = Kolejka MQQueue* **.ProcessName**

Właściwość Nazwa QueueManager

Odczyt-zapis. Nazwa menedżera kolejek produktu IBM MQ .

Zdefiniowane w: Klasa MQQueue

Typ danych: String

Składnia: Aby uzyskać: *QueueManagerNazwa$* = *MQQueue* **.QueueManagerNazwa**

Aby ustawić: *MQQueue* **.QueueManagerName** = *QueueManagerName$*

Właściwość QueueType

Tylko do odczytu. Atrybut QType MQI.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Wartości:

- MQQT_ALIAS
- MQQT_LOCAL
- MODEL MQQT_MODEL
- MQQT_REMOTE

Składnia: Aby uzyskać: *queuetype & = MQQueue* **.QueueType**

Właściwość ReasonCode

Tylko do odczytu. Zwraca kod przyczyny ustawiony przez ostatnią metodę lub dostęp do właściwości wystawiony dla obiektu.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Wartości:

- Patrz Kody zakończenia i przyczyny interfejsu API.

Składnia: do pobrania: *kod_przyczyny & = Kolejka MQQueue* **.ReasonCode**

Właściwość ReasonName

Tylko do odczytu. Zwraca nazwę symboliczną najnowszego kodu przyczyny. Na przykład: "MQRC_QMGR_NOT_AVAILABLE".

Zdefiniowane w: Klasa MQQueue

Typ danych: String

Wartości:

- Patrz Kody zakończenia i przyczyny interfejsu API.

Składnia: aby uzyskać następujące informacje: *reasonname $= MQQueue* **.ReasonName**

Właściwość RemoteQueueManagerName

Tylko do odczytu. Nazwa zdalnego menedżera kolejek. Poprawna tylko w przypadku kolejek zdalnych.

Zdefiniowane w: Klasa MQQueue

Typ danych: Łańcuch o długości 48 znaków

Składnia: Aby uzyskać: *remqmanname $= MQQueue* **.RemoteQueueManagerName**

Właściwość Nazwa RemoteQueueName

Tylko do odczytu. Nazwa kolejki, o której wiadomo, że jest ona znana w zdalnym menedżerze kolejek. Poprawna tylko w przypadku kolejek zdalnych.

Zdefiniowane w: Klasa MQQueue

Typ danych: Łańcuch o długości 48 znaków

Składnia: Aby uzyskać: *remqname $= MQQueue* **.RemoteQueueName**

Właściwość ResolvedQueueManagerName

Tylko do odczytu. Nazwa docelowego menedżera kolejek, który jest znany menedżerowi kolejek lokalnych.

Zdefiniowane w: Klasa MQQueue

Typ danych: Łańcuch o długości 48 znaków

Składnia: aby uzyskać następujące informacje: *resqmanname $= MQQueue* **.ResolvedQueueManagerName**

Właściwość Nazwa ResolvedQueue

Tylko do odczytu. Nazwa docelowej kolejki docelowej, która jest znana menedżerowi kolejek lokalnych.

Zdefiniowane w: Klasa MQQueue

Typ danych: Łańcuch o długości 48 znaków

Składnia: Aby uzyskać: *resqname $= MQQueue* **.ResolvedQueueNazwa**

Właściwość RetentionInterval

Tylko do odczytu. Okres, w którym kolejka powinna zostać zachowana.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Składnia: do pobrania: *retinterval & = MQQueue* **.RetentionInterval**

Właściwość zasięgu

Tylko do odczytu. Określa, czy pozycja dla tej kolejki istnieje również w katalogu komórki.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Wartości:

- MQSCO_Q_MGR
- Komórka MQSCO_CELL

Składnia: do pobrania: *scope & = MQQueue* **.Scope**

Właściwość ServiceInterval

Tylko do odczytu. Atrybut MQI QServiceInterval .

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Składnia: Aby uzyskać: *serviceinterval & = MQQueue.* **ServiceInterval**

Właściwość zdarzenia ServiceIntervalEvent

Tylko do odczytu. Atrybut zdarzenia MQI QServiceInterval.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Wartości:

- MQQSIE_WYSOKI
- MQQSIE_OK
- MQQSIE_NONE

Składnia: Aby uzyskać: *serviceintervalevent & = MQQueue.* **ServiceIntervalZdarzenie**

Właściwość współużytkowalności

Tylko do odczytu. Współużytkowalność kolejki.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Wartości:

- MQQA_SHAREABLE
- MQQA_NOT_SHAREABLE

Składnia: Aby uzyskać następujące informacje: *shareability & = MQQueue* **.Shareability**

Właściwość Nazwa kolejki TransmissionQueue

Tylko do odczytu. Nazwa kolejki transmisji. Poprawna tylko w przypadku kolejek zdalnych.

Zdefiniowane w: Klasa MQQueue

Typ danych: Łańcuch o długości 48 znaków

Składnia: aby uzyskać: *transqname $= MQQueue* **.TransmissionQueueName**

Właściwość TriggerControl

Odczyt-zapis. Sterowanie wyzwalaczem.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Wartości:

- MQTC_OFF
- MQTC_ON

Składnia: Aby uzyskać następujące informacje: *trigcontrol & = MQQueue* **.TriggerControl**

Aby ustawić: *MQQueue* **.TriggerControl** = *trigcontrol &*

Właściwość TriggerData

Odczyt-zapis. Dane wyzwalacza.

Zdefiniowane w: Klasa MQQueue

Typ danych: Łańcuch o długości 64 znaków

Składnia: Aby uzyskać: *trigdata $= MQQueue* **.TriggerData**

Aby ustawić: *MQQueue* **.TriggerData** = *trigdata $*

Właściwość TriggerDepth

Odczyt-zapis. Liczba komunikatów, które muszą znajdować się w kolejce, zanim zostanie zapisany komunikat wyzwalacza.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Składnia: Aby uzyskać: *trigdepth & = MQQueue* **.TriggerDepth**

Aby ustawić: *MQQueue* **.TriggerDepth** = *trigdepth &*

Właściwość priorytetu TriggerMessage

Odczyt-zapis. Priorytet komunikatu progowego dla wyzwalaczy.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Składnia: Aby uzyskać następujące informacje: *trigmesspriority & = MQQueue* **.TriggerMessagePriority**

Aby ustawić wartość: *MQQueue* **.TriggerMessagePriority** = *trigmesspriority &*

Właściwość TriggerType

Odczyt-zapis. Typ wyzwalacza.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Wartości:

- MQTT_NONE
- MQTT_FIRST
- MQTT_EVERY
- MQTT_DEPTH

Składnia: Aby uzyskać: *trigtype & = MQQueue* **.TriggerType**

Aby ustawić wartość: *MQQueue* **.TriggerType** = *Trigtype &*

właściwość użycia

Tylko do odczytu. Wskazuje, dla której jest używana kolejka.

Zdefiniowane w: Klasa MQQueue

Typ danych: Long

Wartości:

- MQUS_NORMAL
- MQUS_TRANSMISSION

Składnia: Aby uzyskać: *usage & = MQQueue* **.Użycie**

ClearError-metoda kodów

Resetuje parametr CompletionCode do wartości MQCC_OK i ReasonCode na wartość MQRC_NONE zarówno dla klasy MQQueue, jak i klasy MQSession.

Zdefiniowane w: Klasa MQQueue

Składnia:

```
Call MQQueue.ClearErrorCodes()
```

Metoda zamknięcia

Zamyka kolejkę przy użyciu bieżących wartości CloseOptions.

Zdefiniowane w: Klasa MQQueue

Składnia:

```
Call MQQueue.Close()
```

metoda GET

Pobiera komunikat z kolejki.

Ta metoda pobiera obiekt MQMessage jako parametr, korzystając z niektórych pól w strukturze MQMD obiektu jako parametry wejściowe. W szczególności używane są pola MessageId i CorrelId , dlatego ważne jest, aby upewnić się, że pola te są ustawione zgodnie z wymaganiami. Więcej informacji na temat tych pól można znaleźć w sekcji MsgId (MQBYTE24) i CorrelId (MQBYTE24).

Jeśli metoda zakończy się niepowodzeniem, obiekt MQMessage nie zostanie zmieniony. Jeśli powiedzie się, porcje MQMD i Message Data w obiekcie MQMessage są zastępowane przez dane MQMD i Message Data z komunikatu przychodzącego. Właściwości elementu sterującego MQMessage są ustawiane w następujący sposób:

- Parametr **MessageLength** jest ustawiony na długość komunikatu IBM MQ .
- Parametr **DataLength** jest ustawiony na długość komunikatu IBM MQ .

- **DataOffset** jest ustawione na zero

Zdefiniowane w:
Klasa MQQueue

Składnia:

```
Call MQQueue.Get(Message, GetMsgOptions, GetMsgLength)
```

Parametry
Komunikat:

Obiekt MQMessage reprezentujący komunikat, który ma zostać pobrany.

Opcje GetMsg:

Opcjonalny obiekt opcji MQGetMessage, który umożliwia sterowanie operacją pobierania. Jeśli ten parametr nie zostanie określony, zostaną użyte domyślne opcje MQGetMessage.

GetMsgDługość:

Opcjonalna wartość 2-lub 4-bajtowa, która umożliwia sterowanie maksymalną długością komunikatu IBM MQ pobranego z kolejki.

Jeśli zostanie podana opcja MQGMO_ACCEPT_TRUNCATED_MSG, operacja GET zakończy się pomyślnie z kodem zakończenia MQCC_WARNING i kodem przyczyny MQRC_TRUNCATED_MSG_ACCEPTED, jeśli wielkość komunikatu przekracza podaną długość.

Obiekt MessageData zawiera pierwsze bajty GetMsg(GetMsg).

Jeśli określono wartość MQGMO_ACCEPT_TRUNCATED_MSG ***nie*** , a wielkość komunikatu przekracza określoną długość, zwracany jest kod zakończenia MQCC_FAILED razem z kodem przyczyny MQRC_TRUNCATED_MESSAGE_FAILED.

Jeśli zawartość buforu komunikatów jest niezdefiniowana, całkowita długość komunikatu jest ustawiana na pełną długość komunikatu, który zostałby pobrany.

Jeśli parametr długości komunikatu nie zostanie określony, długość buforu komunikatów zostanie automatycznie dopasowana do co najmniej wielkości komunikatu przychodzącego.

Metoda otwierania

Otwiera kolejkę przy użyciu bieżących wartości:

1. QueueName
2. QueueManagerName
3. Identyfikator AlternateUser
4. Nazwa DynamicQueue

Zdefiniowane w:
Klasa MQQueue

Składnia:

```
Call MQQueue.Open()
```

metoda PUT

Umieszcza komunikat w kolejce.

Ta metoda przyjmuje obiekt MQMessage jako parametr. Właściwości deskryptora komunikatu (MQMD) tego obiektu mogą zostać zmienione w wyniku tej metody. Wartości, które mają natychmiast po wykonaniu tej metody, to wartości, które zostały wprowadzone do IBM MQ.

Modyfikacje obiektu MQMessage po zakończeniu operacji Put nie mają wpływu na rzeczywisty komunikat w kolejce produktu IBM MQ .

Zdefiniowane w:
Klasa MQQueue

Składnia:

```
Call MQQueue.Put(Message, PutMsgOptions)
```

Parametry
Komunikat

Obiekt MQMessage reprezentujący komunikat, który ma zostać umieszczony.

Opcje PutMsg

MQPutMessageObiekt opcji zawierający opcje służące do sterowania operacją put. Jeśli te wartości nie zostaną określone, zostaną użyte domyślne opcje PutMessage.

Klasa MQMessage

Ta klasa reprezentuje komunikat IBM MQ . Zawiera ona właściwości służące do hermetyzowania deskryptora komunikatu produktu IBM MQ (MQMD) i udostępnia bufor do przechowywania danych komunikatu zdefiniowanych przez aplikację.

Klasa zawiera metody zapisu służące do kopiowania danych z aplikacji ActiveX do obiektu MQMessage. Podobnie klasa zawiera metody odczytu służące do kopiowania danych z obiektu MQMessage do aplikacji ActiveX . Klasa zarządza przydzielaniem i rozdzielaniem pamięci dla buforu automatycznie. Aplikacja nie musi deklarować wielkości buforu, gdy tworzony jest obiekt MQMessage, ponieważ bufor rośnie, aby pomieścić zapisywane do niego dane.

Nie można umieścić komunikatu w kolejce IBM MQ , jeśli wielkość buforu przekracza właściwość MaximumMessageLength tej kolejki.

Po zbudowaniu obiekt MQMessage może zostać umieszczony w kolejce produktu IBM MQ przy użyciu metody MQQueue.Put . Ta metoda pobiera kopię części obiektu MQMD i danych komunikatu obiektu i umieszcza je w kolejce. Aplikacja może w związku z tym zmodyfikować lub usunąć obiekt MQMessage po umieszczonym w kolejce operacji umieszczania, bez wpływu na komunikat w kolejce produktu IBM MQ . Menedżer kolejek może dopasować niektóre pola w strukturze MQMD, gdy kopiuje komunikat w kolejce produktu IBM MQ .

Komunikat przychodzący może zostać odczytany w obiekcie MQMessage za pomocą metody MQQueue.Get . Spowoduje to zastąpienie wszystkich danych MQMD lub danych komunikatu, które mogły już być w obiekcie MQMessage, wartościami z komunikatu przychodzącego. Dopasowuje wielkość buforu danych obiektu MQMessage, tak aby była zgodna z wielkością danych komunikatu przychodzącego.

Zawieranie

Komunikaty są zawarte w klasie MQSession.

Tworzenie

Nowy -tworzy obiekt MQMessage. Jego właściwości deskryptora komunikatu są początkowo ustawiane na wartości domyślne, a bufor danych komunikatu jest pusty.

Składnia

```
Dim msg As New MQMessage
```

lub wersji

```
Set msg = New MQMessage
```

Właściwości

Właściwości elementu sterującego to:

- “Właściwość CompletionCode” na stronie 727
- “Właściwość DataLength” na stronie 727
- “Właściwość DataOffset” na stronie 727
- “Właściwość MessageLength” na stronie 728
- “Właściwość ReasonCode” na stronie 728
- “Właściwość ReasonName” na stronie 728

Właściwości deskryptora komunikatu to:

- “Właściwość AccountingToken” na stronie 728
- “Właściwość Hex AccountingToken” na stronie 729
- “Właściwość danych ApplicationId” na stronie 729
- “Właściwość danych ApplicationOrigin” na stronie 729
- “Właściwość BackoutCount” na stronie 729
- “Właściwość CharacterSet” na stronie 730
- “Właściwość CorrelationId” na stronie 730
- “Właściwość CorrelationIdHex” na stronie 730
- “Właściwość kodowania” na stronie 731
- “Właściwość utraty ważności” na stronie 732
- “Właściwość sprzężenia zwrotnego” na stronie 732
- “Właściwość Format” na stronie 732
- “Właściwość GroupId” na stronie 732
- “Właściwość GroupIdHex” na stronie 732
- “Właściwość MessageData” na stronie 733
- “Właściwość MessageFlags” na stronie 733
- “Właściwość MessageId” na stronie 733
- “Właściwość Hex MessageId” na stronie 734
- “Właściwość Liczba MessageSequence” na stronie 734
- “Właściwość MessageType” na stronie 734
- “Właściwość przesunięcia” na stronie 735
- “Właściwość OriginalLength” na stronie 735
- “Właściwość trwałości” na stronie 735
- “Właściwość priorytetu” na stronie 735
- “PutApplication, właściwość Nazwa” na stronie 735
- “PutApplication, właściwość typu” na stronie 736
- “Właściwość Czas PutDate” na stronie 736
- “Właściwość Nazwa ReplyToQueueManager” na stronie 736
- “Właściwość ReplyToQueueName” na stronie 736
- “Właściwość raportu” na stronie 736
- “Właściwość długości TotalMessageLength” na stronie 737

- “Właściwość UserId” na stronie 737

Metody

- “ClearError-metoda kodów” na stronie 737
- “Metoda ClearMessage” na stronie 737
- “Metoda odczytu” na stronie 738
- “Metoda ReadBoolean” na stronie 738
- “Metoda ReadByte” na stronie 738
- “Metoda ReadDecimal2” na stronie 738
- “Metoda ReadDecimal4” na stronie 738
- “Metoda ReadDouble” na stronie 739
- “Metoda ReadDouble4” na stronie 739
- “Metoda ReadFloat” na stronie 739
- “Metoda ReadInt2” na stronie 740
- “Metoda ReadInt4” na stronie 740
- “Metoda ReadLong” na stronie 740
- “Metoda ReadNullTerminatedString” na stronie 740
- “Metoda ReadShort” na stronie 740
- “Metoda ReadString” na stronie 741
- “Metoda ReadUInt2” na stronie 741
- “ReadUnsigned-metoda typu Byte” na stronie 741
- “Metoda ReadUTF” na stronie 742
- “Metoda ResizeBuffer” na stronie 742
- “Metoda zapisu” na stronie 742
- “Metoda WriteBoolean” na stronie 743
- “Metoda WriteByte” na stronie 743
- “Metoda WriteDecimal2” na stronie 743
- “Metoda WriteDecimal4” na stronie 743
- “Metoda WriteDouble” na stronie 744
- “Metoda WriteDouble4” na stronie 744
- “Metoda WriteFloat” na stronie 744
- “Metoda WriteInt2” na stronie 745
- “Metoda WriteInt4” na stronie 745
- “Metoda WriteLong” na stronie 745
- “Metoda WriteNullTerminatedString” na stronie 745
- “Metoda WriteShort” na stronie 746
- “Metoda WriteString” na stronie 746
- “Metoda WriteUInt2” na stronie 746
- “WriteUnsigned-metoda typu Byte” na stronie 746
- “Metoda WriteUTF” na stronie 747

Dostęp do właściwości

Wszystkie właściwości mogą być odczytane w dowolnym momencie.

Właściwości elementu sterującego są tylko do odczytu, z wyjątkiem DataOffset , które jest odczytywanie_odczytu. Właściwości deskryptora komunikatu to wszystkie operacje odczytu i zapisu, z wyjątkiem BackoutCount i TotalMessage, które są tylko do odczytu.

Należy jednak pamiętać, że niektóre właściwości MQMD mogą być modyfikowane przez menedżer kolejek, gdy komunikat jest umieszczany w kolejce produktu IBM MQ . Szczegółowe informacje na temat sposobu ich modyfikowania znajdują się w polach MQMD .

Konwersja danych

Dane binarne można przekazać do komunikatu IBM MQ , ustawiając właściwość **CharacterSet** w taki sposób, aby była zgodna z identyfikatorem kodowanego zestawu znaków menedżera kolejek (MQCCSI_Q_MGR) i przekazując dane do komunikatu jako łańcuch. Jeśli łańcuch musi zawierać numery kodowe Unicode lub ASCII, można użyć funkcji chr $w celu przekształcenia ich w format łańcuchowy.

Metody Read i Write wykonują konwersję danych. Są one przekształcane między formatami wewnętrznymi ActiveX , a formatami komunikatów produktu IBM MQ zgodnie z definicją właściwości Encoding i CharacterSet z deskryptora komunikatu. Podczas zapisywania komunikatu ustaw wartości w kodowaniu oraz CharacterSet , które są zgodne z charakterystyką odbiorcy komunikatu przed wywołaniem metody Write. Podczas odczytu komunikatu ten krok nie jest zwykle wymagany, ponieważ wartości te zostaną ustawione na podstawie tych wartości w przychodzącym MQMD.

Jest to dodatkowy krok konwersji danych, który jest wykonywany po konwersji wykonanej przez metodę MQQueue.Get .

Właściwość CompletionCode

Tylko do odczytu. Zwraca kod zakończenia IBM MQ ustawiony za pomocą najnowszej metody lub dostępu do właściwości wydanych dla tego obiektu.

Zdefiniowane w: Klasa MQMessage

Typ danych: Long

Wartości:

- MQCC_OK
- MQCC_WARNING,
- MQCC_FAILED

Składnia: Aby uzyskać: *completioncode & = MQMessage* **.CompletionCode**

Właściwość DataLength

Tylko do odczytu. Ta właściwość zwraca wartość:

```
MQMessage.MessageLength - MQMessage.DataOffset
```

Może być używany przed metodą odczytu, aby sprawdzić, czy oczekiwana liczba znaków jest rzeczywiście obecna w buforze.

Wartością początkową jest zero.

Zdefiniowane w: Klasa MQMessage

Typ danych: Long

Składnia: Aby uzyskać informacje: *bytesleft & = MQMessage* **.DataLength**

Właściwość DataOffset

Odczyt-zapis. Bieżąca pozycja w części Dane komunikatu obiektu komunikatu.

Wartość jest wyrażona jako przesunięcie bajtu od początku buforu danych komunikatu. Pierwszy znak w buforze odpowiada wartości DataOffset o wartości zero.

Metoda odczytu lub zapisu rozpoczyna swoją operację od znaku, do którego odwołuje się DataOffset. Te metody przetwarzają dane w buforze sekwencyjnie z tej pozycji i aktualizują wartość DataOffset tak, aby wskazywała na bajt (jeśli istnieje) bezpośrednio po ostatnim przetworzonym bajcie.

Parametr DataOffset może przyjmować tylko wartości z zakresu od zera do MessageLength włącznie. Gdy DataOffset = MessageLength wskazuje na koniec, to jest pierwszy niepoprawny znak buforu. W tej sytuacji dozwolone są metody zapisu-rozszerzają one dane w buforze i zwiększają wartość parametru MessageLength o liczbę dodanych bajtów. Odczyt poza koniec buforu jest niepoprawny.

Wartością początkową jest zero.

Zdefiniowane w: Klasa MQMessage

Typ danych: Long

Składnia: Aby uzyskać: *currpos & = MQMessage* **.DataOffset**

Aby ustawić: *MQMessage* **.DataOffset** = *currpos &*

Właściwość MessageLength

Tylko do odczytu. Zwraca całkowitą długość części danych komunikatu w obiekcie komunikatu, niezależnie od wartości DataOffset.

Wartością początkową jest zero. Jest ona ustawiana na przychodzącą długość komunikatu po wywołaniu metody Get, która odwołuje się do tego obiektu komunikatu. Jest ona zwiększana, jeśli aplikacja korzysta z metody zapisu w celu dodania danych do obiektu. Metody odczytu nie mają wpływu na jego działanie.

Zdefiniowane w: Klasa MQMessage

Typ danych: Long

Składnia: Aby uzyskać informacje: *msglength & = MQMessage* **.MessageLength**

Właściwość ReasonCode

Tylko do odczytu. Zwraca kod przyczyny ustawiony przez najnowszą metodę lub dostęp do właściwości wydany dla tego obiektu.

Zdefiniowane w: Klasa MQMessage

Typ danych: Long

Wartości:

- Patrz Kody zakończenia i przyczyny interfejsu API.

Składnia: do pobrania: *kod_przyczyny & = komunikat MQMessage* **.ReasonCode**

Właściwość ReasonName

Tylko do odczytu. Zwraca nazwę symboliczną najnowszego kodu przyczyny. Na przykład: "MQRC_QMGR_NOT_AVAILABLE". **Zdefiniowane w:** Klasa MQMessage

Typ danych: String

Wartości:

- Patrz Kody zakończenia i przyczyny interfejsu API.

Składnia: aby uzyskać: *reasonname $= MQMessage* **.ReasonName**

Właściwość AccountingToken

Odczyt-zapis. Element MQMD AccountingToken -część kontekstu tożsamości komunikatu.

Jej wartością początkową jest wszystkie wartości NULL.

Zdefiniowane w: Klasa MQMessage

Typ danych: Łańcuch o długości 32 znaków

Składnia: Aby uzyskać: *actoken $= MQMessage* **.AccountingToken**

Ustawienie: *MQMessage.* **AccountingToken** = *actoken $*

Więcej informacji na temat tego, kiedy należy użyć właściwości AccountingTokenHex w miejsce właściwości AccountingToken , zawiera sekcja “Właściwości deskryptora komunikatu” na stronie 687 .

Właściwość Hex AccountingToken

Odczyt-zapis. Element MQMD AccountingToken -część kontekstu tożsamości komunikatu.

Każda z dwóch znaków reprezentuje szesnastkowy odpowiednik pojedynczego znaku ASCII. Na przykład para znaków "6" i "1" oznacza pojedynczy znak "A", a para znaków "6" i "2" reprezentuje pojedynczy znak "B" itd.

Należy podać 64 poprawne znaki szesnastkowe.

Jego wartością początkową jest "0 ... 0"

Zdefiniowane w: Klasa MQMessage

Typ danych: Łańcuch 64 znaków szesnastkowych reprezentujących 32 znaki ASCII

Składnia: Aby uzyskać: *actokenh $= MQMessage* **.AccountingTokenHex**

Ustawienie: *MQMessage.* **AccountingTokenHex** = *actokenh $*

Więcej informacji na temat tego, kiedy należy użyć właściwości AccountingTokenHex w miejsce właściwości AccountingToken , zawiera sekcja “Właściwości deskryptora komunikatu” na stronie 687 .

Właściwość danych ApplicationId

Odczyt-zapis. Element MQMD ApplIdentityData-część kontekstu tożsamości komunikatu.

Jego początkowa wartość to wszystkie odstępy.

Zdefiniowane w: Klasa MQMessage

Typ danych: Łańcuch o długości 32 znaków

Składnia: Aby uzyskać: *applid $= MQMessage* **.ApplicationIdData**

Aby ustawić wartość: *MQMessage* **.ApplicationIdData** = *applid $*

Właściwość danych ApplicationOrigin

Odczyt-zapis. Element danych MQMD ApplOrigin-część kontekstu pochodzenia komunikatu.

Jego początkowa wartość to wszystkie odstępy.

Zdefiniowane w: Klasa MQMessage

Typ danych: Łańcuch o długości 4 znaków

Składnia: aby uzyskać następujące informacje: *applor $= MQMessage* **.ApplicationOriginData**

Aby ustawić wartość: *MQMessage* **.ApplicationOriginData** = *applor $*

Właściwość BackoutCount

Tylko do odczytu. Element MQMD BackoutCount.

Jego wartością początkową jest 0.

Zdefiniowane w: Klasa MQMessage

Typ danych: Long

Składnia: Aby uzyskać następujące informacje: *backoutct & = MQMessage* **.BackoutCount**

Właściwość CharacterSet

Odczyt-zapis. Element MQMD CodedCharSetId.

Jego wartością początkową jest wartość specjalna **MQCCSI_Q_MGR**.

Jeśli parametr CharacterSet jest ustawiony na wartość **MQCCSI_Q_MGR**, strona kodowa dla bieżących ustawień narodowych jest używana do konwersji znaków w metodzie WriteString . W przypadku aplikacji serwera używana strona kodowa jest stroną kodową menedżera kolejek. W przypadku aplikacji klienckich jest to domyślna bieżąca strona kodowa ustawień narodowych.

Na przykład:

```
msg.CharacterSet = MQCCSI_Q_MGR
msg.WriteString(chr$(n))
```

gdzie 'n' jest większe lub równe zeru i mniejsze lub równe 255, powoduje, że jeden bajt wartości' n' jest zapisywany w buforze.

Zdefiniowane w: Klasa MQMessage

Typ danych: Long

Składnia: Aby uzyskać: *:30ccid& = MQMessage* **.CharacterSet**

Aby ustawić: *MQMessage* **.CharacterSet** = *ccid &*

Przykład

Jeśli chcesz, aby łańcuch zapisany na stronie kodowej 437, wywołaj:

```
Message.CharacterSet = 437
Message.WriteString ("string to be written")
```

Przed wywołaniem dowolnego wywołania WriteString ustaw wartość, która ma być ustawiona w CharacterSet .

Właściwość CorrelationId

Odczyt-zapis. Element CorrelationId , który ma zostać włączony do deskryptora MQMD komunikatu podczas umieszczania w kolejce. Identyfikator, który ma być dopasowywany podczas pobierania komunikatu z kolejki.

Jej początkowa wartość jest równa null.

Zdefiniowane w: Klasa MQMessage

Typ danych: Łańcuch o długości 24 znaków

Składnia: Aby uzyskać: *correlid $= MQMessage* **.CorrelationId** , aby ustawić: *MQMessage* **.CorrelationId** *= correlid $*

Aby uzyskać więcej informacji na temat sytuacji, w której należy użyć właściwości CorrelationIdHex w miejsce właściwości CorrelationId , należy zapoznać się z informacjami w sekcji “Właściwości deskryptora komunikatu” na stronie 687 .

Właściwość CorrelationIdHex

Odczyt-zapis. Element CorrelationId , który ma zostać włączony do deskryptora MQMD komunikatu podczas umieszczania w kolejce. Ponadto CorrelationId ma być porównywany podczas pobierania komunikatu z kolejki.

Co dwa znaki w łańcuchu reprezentują szesnastkowy odpowiednik pojedynczego znaku ASCII. Na przykład para znaków "6" i "1" oznacza pojedynczy znak "A", a para znaków "6" i "2" reprezentuje pojedynczy znak "B" itd.

Należy podać 48 poprawnych znaków szesnastkowych.

Jego początkowa wartość to "0 ... 0".

Zdefiniowane w: Klasa MQMessage

Typ danych: Łańcuch o długości 48 znaków szesnastkowych reprezentujących 24 znaki ASCII

Składnia: Aby uzyskać: *correlidh $= MQMessage* **.CorrelationIdHex**

Aby ustawić: *MQMessage* **.CorrelationIdHex** = *correlidh $*

Aby uzyskać informacje na temat sytuacji, w której należy użyć właściwości CorrelationIdHex w miejsce właściwości CorrelationId , należy zapoznać się z programem “Właściwości deskryptora komunikatu” na stronie 687 .

Właściwość kodowania

Odczyt-zapis. Pole MQMD, które identyfikuje reprezentację używaną dla wartości liczbowych w danych komunikatu aplikacji.

Jego wartością początkową jest wartość specjalna MQENC_NATIVE, która różni się w zależności od platformy.

Ta właściwość jest używana przez następujące metody:

- Metoda ReadDecimal2
- Metoda ReadDecimal4
- Metoda ReadDouble
- Metoda ReadDouble4
- Metoda ReadFloat
- Metoda ReadInt2
- Metoda ReadInt4
- Metoda ReadLong
- Metoda ReadShort
- Metoda ReadUInt2
- Metoda WriteDecimal2
- Metoda WriteDecimal4
- Metoda WriteDouble
- Metoda WriteDouble4
- Metoda WriteFloat
- Metoda WriteInt2
- Metoda WriteInt4
- Metoda WriteLong
- Metoda WriteShort
- Metoda WriteUInt2

Zdefiniowane w: Klasa MQMessage

Typ danych: Long

Składnia: Aby uzyskać: *encoding & = MQMessage* **.Kodowanie** Do ustawienia: *MQMessage* **.Kodowanie** = *kodowanie &*

W przypadku przygotowywania do zapisu danych w buforze komunikatów należy ustawić to pole w taki sposób, aby były zgodne z charakterystyką platformy odbierającej menedżera kolejek, jeśli odbierający menedżer kolejek nie jest w stanie wykonać własnej konwersji danych.

Właściwość utraty ważności

Odczyt-zapis. Pole czasu utraty ważności MQMD, oczekiwane w dziesiątych częściach sekundy.

Jej wartością początkową jest wartość specjalna MQEI_UNLIMITED

Zdefiniowane w: Klasa MQMessage

Typ danych: Long

Składnia: Aby uzyskać: *wygaśnięcie & = komunikat MQMessage* **.Utrata ważności**

Aby ustawić: *MQMessage* **.Utrata ważności** = *utrata ważności &*

Właściwość sprzężenia zwrotnego

Odczyt-zapis. Pole informacji zwrotnej MQMD.

Jego początkowa wartość jest wartością specjalną MQFB_NONE.

Zdefiniowane w: Klasa MQMessage

Typ danych: Long

Wartości:

- Patrz Feedback(Opinia).

Składnia: Aby uzyskać: *feedback & = MQMessage* **.Opinia**

Aby ustawić: *MQMessage* **.Opinia** = *informacja zwrotna &*

Właściwość Format

Odczyt-zapis. Pole formatu MQMD. Podaje nazwę wbudowanego lub zdefiniowanego przez użytkownika formatu opisowego, który opisuje rodzaj danych komunikatu.

Jego wartością początkową jest wartość specjalna MQFMT_NONE.

Zdefiniowane w: Klasa MQMessage

Typ danych: Łańcuch o długości 8 znaków

Składnia: Aby uzyskać: *format $= MQMessage* **.Format**

Aby ustawić: *MQMessage* **.Format** = *format $*

Właściwość GroupId

Odczyt-zapis. Element GroupId , który ma zostać włączony do rekordu MQPMR komunikatu podczas umieszczania w kolejce. Identyfikator, który ma być dopasowywany podczas pobierania komunikatu z kolejki. Jej wartością początkową jest wszystkie wartości NULL.

Zdefiniowane w:
Klasa MQMessage

Typ danych:
Łańcuch 24 znaków

Składnia: Aby uzyskać: *groupid $= MQMessage*. **GroupId**

Aby ustawić wartość: *MQMessage*. **GroupId** = *groupid $*

Aby uzyskać więcej informacji na temat sytuacji, w której należy użyć wartości GroupIdHex w miejsce właściwości GroupId , należy zapoznać się z informacjami w sekcji “Właściwości deskryptora komunikatu” na stronie 687 .

Właściwość GroupIdHex

Odczyt-zapis. Element GroupId , który ma zostać włączony do rekordu MQPMR komunikatu podczas umieszczania w kolejce. Identyfikator, który ma być dopasowywany podczas pobierania komunikatu z kolejki.

Co dwa znaki w łańcuchu reprezentują szesnastkowy odpowiednik pojedynczego znaku ASCII. Na przykład para znaków "6" i "1" oznacza pojedynczy znak "A", a para znaków "6" i "2" reprezentuje pojedynczy znak "B" itd.

Należy podać 48 poprawnych znaków szesnastkowych.

Jego początkowa wartość to "0 ... 0".

Zdefiniowane w:
Klasa MQMessage

Typ danych:
Łańcuch 48 znaków szesnastkowych, reprezentujący 24 znaki ASCII.

Składnia: Aby uzyskać: *groupidh $= MQMessage*. **GroupIdHex**

Aby ustawić wartość: *MQMessage*. **GroupIdHex** = *groupidh $*

Aby uzyskać więcej informacji na temat sytuacji, w której należy użyć wartości GroupIdHex w miejsce właściwości GroupId , należy zapoznać się z informacjami w sekcji “Właściwości deskryptora komunikatu” na stronie 687 .

Właściwość MessageData

Odczyt-zapis. Pobiera lub ustawia całą zawartość komunikatu jako łańcuch znaków.

Zdefiniowane w: Klasa MQMessage

Typ danych: Variant

Uwaga: Typ danych używany przez tę właściwość jest typu Variant, ale MQAX oczekuje, że będzie to typ wariantu typu String. Jeśli zostanie przekazany wariant innego typu niż ten typ, zostanie zwrócony błąd MQRC_OBJECT_TYPE_ERROR.

Składnia: aby uzyskać: *String$* = *MQMessage* **.MessageData**

Aby ustawić: *MQMessage* **.MessageData** = *String$*

Właściwość MessageFlags

Odczyt-Zapis. Flagi komunikatów określające informacje sterujące Segmentation. Wartością początkową jest 0.

Zdefiniowane w:
Klasa MQMessage

Typ danych:
Długa liczba całkowita

Wartości:
Patrz MsgFlags (MQLONG).

Składnia: Aby uzyskać informacje: *messageflags & = MQMessage*. **MessageFlags**

Aby ustawić wartość: *MQMessage*. **MessageFlags** = *aglaga_komunikatów &*

Właściwość MessageId

Odczyt-zapis. Element MessageId , który ma zostać włączony do deskryptora MQMD komunikatu podczas umieszczania w kolejce. Identyfikator, który ma być dopasowywany podczas pobierania komunikatu z kolejki.

Jej wartością początkową jest wszystkie wartości NULL.

Zdefiniowane w: Klasa MQMessage

Typ danych: Łańcuch o długości 24 znaków

Składnia: Aby uzyskać następujący komunikat: *messageid $= MQMessage* **.MessageId**

Aby ustawić: *MQMessage* **.MessageId** = *messageid $*

Aby uzyskać więcej informacji na temat sytuacji, w której należy użyć wartości MessageIdHex w miejsce właściwości MessageId , należy zapoznać się z informacjami w sekcji “Właściwości deskryptora komunikatu” na stronie 687 .

Właściwość Hex MessageId

Odczyt-zapis. Element MessageId , który ma zostać włączony do deskryptora MQMD komunikatu podczas umieszczania w kolejce. Ponadto MessageId ma być porównywany podczas pobierania komunikatu z kolejki.

Co dwa znaki w łańcuchu reprezentują szesnastkowy odpowiednik pojedynczego znaku ASCII. Na przykład para znaków "6" i "1" oznacza pojedynczy znak "A", a para znaków "6" i "2" reprezentuje pojedynczy znak "B" itd.

Należy podać 48 poprawnych znaków szesnastkowych.

Jego początkowa wartość to "0 ... 0".

Zdefiniowane w: Klasa MQMessage

Typ danych: Łańcuch o długości 48 znaków szesnastkowych reprezentujących 24 znaki ASCII

Składnia: Aby uzyskać: *messageidh $= MQMessage* **.MessageIdHex**

Aby ustawić: *MQMessage* **.MessageIdHex** = *messageidh $*

Aby uzyskać więcej informacji na temat sytuacji, w której należy użyć wartości MessageIdHex w miejsce właściwości MessageId , należy zapoznać się z informacjami w sekcji “Właściwości deskryptora komunikatu” na stronie 687 .

Właściwość Liczba MessageSequence

Odczyt-Zapis. Informacje o sekwencji identyfikujące komunikat w grupie. Wartością początkową jest 1.

Zdefiniowane w:
Klasa MQMessage

Typ danych:
Długa liczba całkowita

Wartości:
Patrz MsgSeqNumber (MQLONG).

Składnia: Aby uzyskać następujący komunikat: *sequencenumber & = MQMessage*. **SequenceNumber**

Aby ustawić wartość: *MQMessage*. **SequenceNumber** = *sequencenumber &*

Właściwość MessageType

Odczyt-zapis. Pole MsgType deskryptora MQMD.

Jego początkowa wartość to MQMT_DATAGRAM.

Zdefiniowane w: Klasa MQMessage

Typ danych: Long

Wartości:

- Patrz MsgType (MQLONG).

Składnia: do pobrania: *msgtype & = MQMessage* **.MessageType**

Aby ustawić: *MQMessage* **.MessageType** = *typ_komunikatu &*

Właściwość przesunięcia

Odczyt-Zapis. Przesunięcie w posegmentowanym komunikacie. Wartością początkową jest 0.

Zdefiniowane w:
Klasa MQMessage

Typ danych:
Długa liczba całkowita

Wartości:
Patrz Przesunięcie (MQLONG).

Składnia: Do pobrania: *przesunięcie & = komunikat MQMessage*. **Depozycja**

Aby ustawić wartość: *MQMessage*. **Przesunięcie** = *przesunięcie &*

Właściwość OriginalLength

Odczyt-Zapis. Oryginalna długość segmentowanego komunikatu. Wartością początkową jest MQOL_UNDEFINED.

Zdefiniowane w:
Klasa MQMessage

Typ danych:
Długa liczba całkowita

Wartości:
Patrz OriginalLength (MQLONG).

Składnia: Aby uzyskać: *originallength & = MQMessage*. **OriginalLength**

Aby ustawić wartość: *MQMessage*. **OriginalLength** = *originallength &*

Właściwość trwałości

Odczyt-zapis. Ustawienie trwałości komunikatu.

Jej wartością początkową jest MQPER_PERSISTENCE_AS_Q_DEF.

Zdefiniowane w: Klasa MQMessage

Typ danych: Long

Składnia: Aby uzyskać: *utrwal & = komunikat_MQMessage* **.Trwałość**

Aby ustawić: *MQMessage* **.Trwałość** = *utrwal &*

Właściwość priorytetu

Odczyt-zapis. Priorytet komunikatu.

Jej wartością początkową jest wartość specjalna MQPRI_PRIORITY_AS_Q_DEF

Zdefiniowane w: Klasa MQMessage

Typ danych: Long

Składnia: Aby uzyskać: *priority & = MQMessage* **.Priorytet**

Aby ustawić: *MQMessage* **.Priorytet** = *priorytet &*

PutApplication, właściwość Nazwa

Odczyt-zapis. Nazwa wywołania MQMD PutAppl-część kontekstu źródłowego komunikatu.

Jego początkowa wartość to wszystkie odstępy.

Zdefiniowane w: Klasa MQMessage

Typ danych: Łańcuch o długości 28 znaków

Składnia: Aby uzyskać: *putapplnm $= MQMessage* **.PutApplicationNazwa**

Aby ustawić wartość: *MQMessage* **.PutApplicationName** = *putapplnm $*

PutApplication, właściwość typu

Odczyt-zapis. Typ wywołania MQMD PutAppl-część kontekstu źródłowego komunikatu.

Jego początkowa wartość to MQAT_NO_CONTEXT

Zdefiniowane w: Klasa MQMessage

Typ danych: Long

Wartości:

- Patrz PutApplType (MQLONG).

Składnia: Aby uzyskać: *putappltp & = MQMessage* **.PutApplicationTyp**

Aby ustawić wartość: *MQMessage* **.PutApplicationType** = *putappltp &*

Właściwość Czas PutDate

Odczyt/zapis. Ta właściwość łączy pola PutDate (PutDate) i PutTime (PutTime) z tabeli MQMD. Są to części kontekstu źródłowego komunikatu, które wskazują, kiedy komunikat został umieszczony.

Rozszerzenie ActiveX jest przekształcane między formatem daty/godziny ActiveX , a formatami daty i godziny używani w strukturze MQMD produktu IBM MQ . Jeśli zostanie odebrany komunikat, który ma niepoprawną wartość PutDate lub PutTime, właściwość Czas PutDateTime po metodzie get zostanie ustawiona na wartość EMPTY.

Jego wartością początkową jest EMPTY.

Zdefiniowane w: Klasa MQMessage

Typ danych: Variant typu 7 (data/godzina) lub EMPTY.

Składnia: Aby uzyskać: *datetime = MQMessage* **.PutDate**

Aby ustawić: *MQMessage* **.PutDate, godzina** = *datetime*

Właściwość Nazwa ReplyToQueueManager

Odczyt-zapis. Pole menedżera kolejek MQMD ReplyTo.

Wartość początkowa to wszystkie odstępy.

Zdefiniowane w: Klasa MQMessage

Typ danych: Łańcuch o długości 48 znaków

Składnia: Aby uzyskać: *replytoqmgr $= MQMessage* **.ReplyToQueueManagerName**

Aby ustawić wartość: *MQMessage* **.ReplyToQueueManagerName** = *replytoqmgr $*

Właściwość ReplyToQueueName

Odczyt-zapis. Pole Q MQMD ReplyTo(ReplyTo).

Wartość początkowa to wszystkie odstępy.

Zdefiniowane w: Klasa MQMessage

Typ danych: Łańcuch o długości 48 znaków

Składnia: Do pobrania: *replytoq $= MQMessage* **.ReplyToQueueName**

Aby ustawić: *MQMessage* **.ReplyToQueueName** = *replytoq $*

Właściwość raportu

Odczyt-zapis. Opcje raportu.

Jego początkowa wartość to MQRO_NONE.

Zdefiniowane w: Klasa MQMessage

Typ danych: Long

Wartości:

- Patrz Raport.

Składnia: Aby uzyskać: *report & = MQMessage* **.Raport**

Aby ustawić: *MQMessage* **.Raport** = *raport &*

Właściwość długości TotalMessageLength

Tylko do odczytu. Pobiera długość ostatniego komunikatu odebranego przez komendę MQGET. Jeśli komunikat nie został obcięty, ta wartość jest równa wartości właściwości MessageLength .

Zdefiniowane w: Klasa MQMessage

Typ danych: Long

Składnia: Aby uzyskać informacje: *totalmessagelength & = MQMessage* **.TotalMessageLength**

Właściwość UserId

Odczyt-zapis. Element MQMD UserIdentifier -część kontekstu tożsamości komunikatu.

Jego początkowa wartość to wszystkie odstępy.

Zdefiniowane w: Klasa MQMessage

Typ danych: Łańcuch o długości 12 znaków

Składnia: Aby uzyskać: *userid $= MQMessage* **.UserId**

Aby ustawić: *MQMessage* **.UserId** = *id_użytkownika $*

ClearError-metoda kodów

Resetuje parametr CompletionCode do wartości MQCC_OK i ReasonCode na wartość MQRC_NONE zarówno dla klasy MQMessage, jak i klasy MQSession.

Zdefiniowane w: Klasa MQMessage

Składnia:

```
Call MQMessage.ClearErrorCodes()
```

Metoda ClearMessage

Ta metoda kasuje część buforu danych obiektu MQMessage. Wszystkie dane komunikatu w buforze danych zostaną utracone, ponieważ wartości MessageLength, DataLengthi DataOffset są ustawione na zero.

Część deskryptora komunikatu (MQMD) nie może zostać zmieniona; przed ponownym użyciem obiektu MQMessage może być konieczna modyfikacja niektórych pól MQMD. Aby ponownie ustawić pola MQMD, użyj opcji Nowa, aby zastąpić obiekt nową instancją.

Zdefiniowane w: Klasa MQMessage

Składnia:

```
Call MQMessage.ClearMessage()
```

Metoda odczytu

Odczytuje sekwencję bajtów z buforu komunikatów do tablicy bajtów. Wartość DataOffset jest zwiększana, a długość danych jest zmniejszana o liczbę odczytanych bajtów.

Zdefiniowane w:
Klasa MQMessage

Składnia: Dane = MQMessage. **Odczyt** (len &)

Parametry:
len &: Long. Długość danych w bajtach do odczytania.

Metoda ReadBoolean

Odczytuje jednobajtową wartość boolowskową z bieżącej pozycji w buforze komunikatów i zwraca 2-bajtową wartość boolowskim TRUE (-1) /FALSE (0). Wartość DataOffset jest zwiększana o jeden, a długość danych jest zmniejszana o jeden.

Zdefiniowane w:
Klasa MQMessage

Składnia: *value = MQMessage*. **ReadBoolean**

Metoda ReadByte

Począwszy od bajtu, do którego odwołuje się funkcja DataOffset, metoda ReadByte odczytuje 1 bajt z buforu danych komunikatów i zwraca ją jako liczbę całkowitą (2-bajtową ze znakiem) całkowitą z zakresu od -128 do 127.

Metoda kończy się niepowodzeniem, jeśli właściwość MQMessage.DataLength jest mniejsza niż 1, gdy jest ona wydawana.

Wartość DataOffset jest zwiększana o 1, a DataLength jest zmniejszana o 1, jeśli metoda zakończy się powodzeniem.

Przyjmuje się, że bajt danych komunikatu jest binarną liczbą całkowitą ze znakiem.

Zdefiniowane w:
Klasa MQMessage

Składnia:
integerv% = MQMessage **.ReadByte**

Metoda ReadDecimal2

Odczytuje 2-bajtową upakowaną liczbę dziesiętną i zwraca ją jako dwubajtową liczbę całkowitą ze znakiem. Wartość DataOffset jest zwiększana o dwie, a długość danych jest zmniejszana o dwa.

Zdefiniowane w:
Klasa MQMessage

Składnia: *wartość% = komunikat MQMessage*. **ReadDecimal2**

Metoda ReadDecimal4

Odczytuje 4-bajtowy upakowany numer dziesiętny i zwraca go jako 4-bajtową liczbę całkowitą ze znakiem. Wartość DataOffset jest zwiększana o cztery, a długość danych jest zmniejszana o cztery.

Zdefiniowane w:
Klasa MQMessage

Składnia:

```
Call value& = MQMessage.ReadDecimal4
```

Metoda ReadDouble

Począwszy od bajtu, do którego odwołuje się funkcja DataOffset, metoda ReadDouble odczytuje 8 bajtów z buforu danych komunikatów i zwraca je jako wartość zmiennopozycyjną typu Double (signed 8-byte).

Metoda kończy się niepowodzeniem, jeśli właściwość MQMessage.DataLength jest mniejsza niż 8, gdy jest ona wydawana.

Wartość DataOffset jest zwiększana o 8, a DataLength jest zmniejszana o 8, jeśli metoda zakończy się powodzeniem.

Przyjmuje się, że 8 znaków danych komunikatu jest binarną liczbą zmiennopozycyjną. Kodowanie jest określane przez właściwość MQMessage.Encoding . Należy pamiętać, że konwersja z formatu System/360 nie jest obsługiwana.

Zdefiniowane w:
Klasa MQMessage

Składnia:
wątpliwo# = MQMessage **.ReadDouble**

Metoda ReadDouble4

Metody ReadDouble4 i WriteDouble4 są alternatywami dla opcji ReadFloat i WriteFloat. Jest to spowodowane tym, że obsługują one 4-bajtowe wartości komunikatów zmiennopozycyjnych System/390 , które są zbyt duże, aby można było przekształcić je w 4-bajtowy format zmiennopozycyjny IEEE.

Począwszy od bajtu, do którego odwołuje się funkcja DataOffset, metoda ReadDouble4 odczytuje 4 bajty z buforu danych komunikatów i zwraca je jako wartość zmiennopozycyjną typu Double (signed 8-byte).

Metoda kończy się niepowodzeniem, jeśli właściwość MQMessage.DataLength jest mniejsza niż 4, gdy jest ona wydawana.

Wartość DataOffset jest zwiększana o 4, a DataLength jest zmniejszana o 4, jeśli metoda zakończy się powodzeniem.

Przyjmuje się, że 4 znaki danych komunikatu są binarną liczbą zmiennopozycyjną. Kodowanie jest określane przez właściwość MQMessage.Encoding . Należy pamiętać, że konwersja z formatu System/360 nie jest obsługiwana.

Zdefiniowane w:
Klasa MQMessage

Składnia:
wątpliwo# = komunikat_MQMessage. **ReadDouble4**

Metoda ReadFloat

Począwszy od bajtu, do którego odwołuje się funkcja DataOffset, metoda ReadFloat odczytuje 4 bajty z buforu danych komunikatów i zwraca je jako wartość zmiennopozycyjną o pojedynczej (podpisanej 4-bajtowej) wartości zmiennopozycyjnej.

Metoda kończy się niepowodzeniem, jeśli właściwość MQMessage.DataLength jest mniejsza niż 4, gdy jest ona wydawana.

Wartość DataOffset jest zwiększana o 4, a DataLength jest zmniejszana o 4, jeśli metoda zakończy się powodzeniem.

Przyjmuje się, że 4 znaki danych komunikatu są liczbą zmiennopozycyjną. Kodowanie jest określane przez właściwość MQMessage.Encoding . Należy pamiętać, że konwersja z formatu System/360 nie jest obsługiwana.

Zdefiniowane w:
Klasa MQMessage

Składnia:
singlev! = MQMessage **.ReadFloat**

Metoda ReadInt2

Metoda jest identyczna z metodą ReadShort .

Składnia:
integerv% = MQMessage **.ReadInt2**

Metoda ReadInt4

Ta metoda jest identyczna z metodą ReadLong .

Składnia:
bigint & = MQMessage **.ReadInt4**

Metoda ReadLong

Począwszy od bajtu, do którego odwołuje się funkcja DataOffset, metoda ReadLong odczytuje 4 bajty z buforu danych komunikatów i zwraca je jako liczbę całkowitą typu Long (podpisaną 4-bajtową).

Metoda kończy się niepowodzeniem, jeśli właściwość MQMessage.DataLength jest mniejsza niż 4, gdy jest ona wydawana.

Wartość DataOffset jest zwiększana o 4, a DataLength jest zmniejszana o 4, jeśli metoda zakończy się powodzeniem.

Przyjmuje się, że 4 znaki danych komunikatu są binarną liczbą całkowitą. Kodowanie jest określane przez właściwość MQMessage.Encoding .

Zdefiniowane w:
Klasa MQMessage

Składnia:
bigint & = MQMessage **.ReadLong**

Metoda ReadNullTerminatedString

Ta metoda jest używana w miejsce ReadString , jeśli łańcuch może zawierać znaki o kodzie zero osadzonym.

Ta metoda odczytuje określoną liczbę bajtów z buforu danych komunikatów, począwszy od bajtu, do którego odwołuje się funkcja DataOffset , i zwraca go jako łańcuch ActiveX . Jeśli łańcuch zawiera osadzoną wartość NULL przed końcem, długość zwróconego łańcucha zostanie zmniejszona tak, aby odzwierciedlała tylko te znaki przed wartością NULL.

Wartość DataOffset jest zwiększana, a wartość DataLength jest zmniejszana o podaną wartość, niezależnie od tego, czy łańcuch zawiera osadzone znaki o wartości NULL.

Zakłada się, że znaki w danych komunikatu są łańcuchami na stronie kodowej, która jest określona przez właściwość MQMessage.CharacterSet . Konwersja na reprezentację ActiveX jest wykonywana dla aplikacji.

Zdefiniowane w:
Klasa MQMessage

Składnia: *łańcuch $ = komunikat MQMessage*. **ReadNullTerminatedString(długość &)**

Parametry:
length & *Long.* Długość pola łańcucha w bajtach.

Metoda ReadShort

Począwszy od bajtu, do którego odwołuje się funkcja DataOffset, metoda ReadShort odczytuje 2 bajty z buforu danych komunikatów i zwraca je jako wartość typu Integer (podpisaną 2-bajtową).

Metoda kończy się niepowodzeniem, jeśli właściwość MQMessage.DataLength jest mniejsza niż wartość 2, gdy jest ona wydawana.

Wartość DataOffset jest zwiększana o 2, a wartość DataLength jest zmniejszana o 2, jeśli metoda powiedzie się.

Przyjmuje się, że 2 znaki danych komunikatu są binarną liczbą całkowitą. Kodowanie jest określane przez właściwość MQMessage.Encoding .

Zdefiniowane w:
Klasa MQMessage

Składnia:
integerv% = MQMessage **.ReadShort**

Metoda ReadString

Ta metoda odczytuje n bajtów z buforu danych komunikatu, począwszy od bajtu, do którego odwołuje się element DataOffset , i zwraca go jako łańcuch ActiveX .

Metoda kończy się niepowodzeniem, jeśli właściwość MQMessage.DataLength jest mniejsza niż n, gdy jest ona wydawana.

Wartość DataOffset jest zwiększana o n, a DataLength jest zmniejszana o n, jeśli metoda zakończy się powodzeniem.

Przyjmuje się, że n znaków danych komunikatu jest łańcuchem na stronie kodowej, który jest określony przez właściwość MQMessage.CharacterSet . Konwersja na reprezentację ActiveX jest wykonywana dla aplikacji.

Zdefiniowane w: Klasa MQMessage

Składnia: *stringv $= MQMessage* **.ReadString** *(długość &*)

Parametr
długość & Długa. Długość pola łańcucha w bajtach.

Metoda ReadUInt2

Począwszy od bajtu, do którego odwołuje się funkcja DataOffset, metoda ReadUInt2 odczytuje 2 bajty z buforu danych komunikatów i zwraca je jako wartość typu Long (podpisaną 4-bajtową).

Metoda kończy się niepowodzeniem, jeśli właściwość MQMessage.DataLength jest mniejsza niż wartość 2, gdy jest ona wydawana.

Wartość DataOffset jest zwiększana o 2, a wartość DataLength jest zmniejszana o 2, jeśli metoda powiedzie się.

Przyjmuje się, że 2 bajty danych komunikatu są binarną liczbą całkowitą bez znaku. Kodowanie jest określane przez właściwość MQMessage.Encoding .

Zdefiniowane w:
Klasa MQMessage

Składnia:
bigint & = MQMessage **.ReadUInt2**

ReadUnsigned-metoda typu Byte

Począwszy od bajtu, do którego odwołuje się funkcja DataOffset, metoda ReadUnsignedodczytuje 1 bajt z buforu danych komunikatów i zwraca ją jako liczbę całkowitą (2-bajtową ze znakiem) całkowitą z zakresu od 0 do 255.

Metoda kończy się niepowodzeniem, jeśli właściwość MQMessage.DataLength jest mniejsza niż 1, gdy jest ona wydawana.

Wartość DataOffset jest zwiększana o 1, a DataLength jest zmniejszana o 1, jeśli metoda zakończy się powodzeniem.

Przyjmuje się, że 1 znak danych komunikatu jest binarną liczbą całkowitą bez znaku.

Zdefiniowane w:
Klasa MQMessage

Składnia:
integerv% = MQMessage **.ReadUnsignedBajt**

Metoda ReadUTF

Ta metoda odczytuje łańcuch formatu UTF z komunikatu, rozpoczynając od bajtu, do którego odwołuje się produkt **DataOffset**, i zwraca łańcuch formatu UTF jako łańcuch ActiveX . Odczytany łańcuch formatu UTF składa się z 2 bajtów danych, które wskazują długość łańcucha, po którym następuje dane znakowe UTF.

Metoda kończy się niepowodzeniem, jeśli wartość **MQMessage.DataLength** jest mniejsza niż długość łańcucha, gdy jest ona wydawana.

Wartość **DataOffset** jest zwiększana o długość łańcucha, a łańcuch **DataLength** jest zmniejszany o długość łańcucha, jeśli metoda zakończy się powodzeniem.

Zdefiniowane w:
Klasa MQMessage

Składnia:

```
value$ = MQMessage.ReadUTF
```

Metoda ResizeBuffer

Ta metoda zmienia ilość pamięci masowej obecnie przydzielonej wewnętrznie w celu przechowywania buforu danych komunikatu. Daje on aplikacji pewną kontrolę nad automatycznym zarządzaniem buforami, w tym, że jeśli aplikacja wie, że ma zajmować się dużym komunikatem, może zapewnić, że przydzielany jest wystarczająco duży bufor. Aplikacja nie musi korzystać z tego wywołania-jeśli tak się nie stanie, automatyczny kod zarządzania buforami powiększy rozmiar bufora, aby zmieścić się.

Jeśli wielkość buforu zostanie ponownie zmniejszona, a bieżąca wartość parametru MessageLengthzostanie zmniejszona, ryzyko utraty danych będzie ryzykować. Jeśli dane zostaną utracone, metoda zwraca kod CompletionCode o wartości MQCC_WARNING i ReasonCode o wartości MQRC_DATA_OBCIĘTE.

W przypadku zmiany wielkości buforu na wartość mniejszą niż wartość właściwości **DataOffset** , należy:

- Właściwość **DataOffset** jest zmieniana w taki sposób, aby wskazywała na koniec nowego buforu.
- Właściwość **DataLength** jest ustawiona na zero.
- Właściwość **MessageLength** została zmieniona na nową wielkość buforu.

Zdefiniowane w:
Klasa MQMessage

Składnia: *MQMessage* **.ResizeBuffer** (*Length &*)

parametr:
Długość & długa. Wielkość wymagana w znakach.

Metoda zapisu

Zapisuje sekwencję bajtów w buforze komunikatów z tablicy bajtów na pozycji, do której odnosi się przesunięcie danych. Jeśli jest to konieczne, długość buforu (MQMessage.MQMessageLength) jest rozszerzana w taki sposób, aby pomieścić pełną długość tablicy bajtów. DataOffset jest zwiększane o liczbę bajtów zapisanych, jeśli metoda zakończy się powodzeniem.

Zdefiniowane w:
Klasa MQMessage

Składnia:

```
Call MQMessage.Write(value)
```

Parametry:
data: tablica bajtów lub odwołanie wariantu do tablicy bajtów

Metoda WriteBoolean

Zapisuje jednobajtową wartość boolowką w bieżącym położeniu w buforze komunikatów z dwubajtowej wartości boolowskiej. Wartość DataOffset jest zwiększana o jeden.

Zdefiniowane w:
Klasa MQMessage

Składnia:

```
Call MQMessage.WriteBoolean(value)
```

parametr:
wartość: Boolean (2-bajty). Wartość, która ma zostać zapisana.

Metoda WriteByte

Ta metoda przyjmuje podpisaną 2-bajtową liczbę całkowitą i zapisuje ją w buforze danych komunikatu jako jednobajtowy numer binarny w położeniu określonym przez wartość DataOffset. Zastępuje on wszystkie dane już znajdujące się w pozycji w buforze i rozszerza długość buforu (MQMessage.MessageLength), jeśli jest to konieczne.

Wartość DataOffset jest zwiększana o jeden, jeśli metoda zakończy się powodzeniem.

Podana wartość powinna miewać się w zakresie od -128 do 127. Jeśli tak nie jest, metoda zwraca wartość CompletionCode MQCC_FAILED i ReasonCode MQRC_WRITE_VALUE_ERROR.

Zdefiniowane w: Klasa MQMessage

Składnia:

```
Call MQMessage.WriteByte(value%)
```

Parametr: *wartość%* Liczba całkowita. Wartość, która ma zostać zapisana.

Metoda WriteDecimal2

Zapisuje podpisaną 2-bajtową liczbę całkowitą w postaci dwubajtowej upakowanej liczby dziesiętnej. Wartość DataOffset jest zwiększana o dwa.

Zdefiniowane w:
Klasa MQMessage

Składnia:

```
Call MQMessage.WriteDecimal2(value%)
```

parametr:
wartość% Integer. Wartość, która ma zostać zapisana.

Metoda WriteDecimal4

Zapisuje podpisaną 4-bajtową liczbę całkowitą jako 4-bajtową upakowaną liczbę dziesiętną. Wartość DataOffset jest zwiększana o cztery.

Zdefiniowane w:
Klasa MQMessage

Składnia:

```
Call MQMessage.WritedDecimal4(value&)
```

parametr:
value & Long. Wartość, która ma zostać zapisana.

Metoda WriteDouble

Ta metoda przyjmuje 8-bajtową wartość zmiennopozycyjną i zapisuje ją w buforze danych komunikatów jako 8-bajtowa liczba zmiennopozycyjna, począwszy od pozycji, do której odwołuje się DataOffset. Zastępuje on wszystkie dane znajdujące się już na tych pozycjach w buforze i rozszerza długość buforu (MQMessage.MessageLength), jeśli jest to konieczne.

Wartość DataOffset jest zwiększana o 8, jeśli metoda zakończy się powodzeniem.

Metoda przekształca się w reprezentację zmiennopozycyjną określoną przez właściwość MQMessage.Encoding . *Konwersja do formatu System/360 nie jest obsługiwana.*

Zdefiniowane w: Klasa MQMessage

Składnia:

```
Call MQMessage.WriteDouble(value#)
```

parametr:
value# Double. Wartość, która ma zostać zapisana.

Metoda WriteDouble4

Opis sytuacji, w których należy użyć funkcji ReadDouble4 i WriteDouble4 w miejsce ReadFloat i WriteFloat, znajduje się w sekcji “Metoda ReadDouble4” na stronie 739 .

Ta metoda przyjmuje 8-bajtową wartość zmiennopozycyjną i zapisuje ją w buforze danych komunikatów jako 4-bajtową liczbę zmiennopozycyjną, rozpoczynając od pozycji, do której odwołuje się DataOffset.

Wartość DataOffset jest zwiększana o 4, jeśli metoda zakończy się powodzeniem.

Zastępuje on wszystkie dane znajdujące się już na tych pozycjach w buforze i rozszerza długość buforu (MQMessage.MessageLength), jeśli jest to konieczne.

Metoda przekształca się w reprezentację zmiennopozycyjną określoną przez właściwość MQMessage.Encoding . *Konwersja do formatu System/360 nie jest obsługiwana.*

Zdefiniowane w: Klasa MQMessage

Składnia:

```
Call MQMessage.WriteDouble4(value#)
```

Parametr: *value#* Double. Wartość, która ma zostać zapisana.

Metoda WriteFloat

Ta metoda przyjmuje 4-bajtową wartość zmiennopozycyjną i zapisuje ją w buforze danych komunikatu jako 4-bajtowy numer zmiennopozycyjny rozpoczynający się od znaku, do którego odwołuje się element DataOffset. Zastępuje on wszystkie dane znajdujące się już na tych pozycjach w buforze i rozszerza długość buforu (MQMessage.MessageLength), jeśli jest to konieczne.

Wartość DataOffset jest zwiększana o 4, jeśli metoda zakończy się powodzeniem.

Metoda przekształca się w reprezentację binarną określoną przez właściwość MQMessage.Encoding . *Konwersja do formatu System/360 nie jest obsługiwana.*

Zdefiniowane w: Klasa MQMessage

Składnia:

```
Call MQMessage.WriteFloat(value!)
```

Parametr *wartość!* Float. Wartość, która ma zostać zapisana.

Metoda WriteInt2

Ta metoda jest identyczna z metodą WriteShort .

Składnia:

```
Call MQMessage.WriteInt2(value%)
```

Parametr *wartość%* Liczba całkowita. Wartość, która ma zostać zapisana.

Metoda WriteInt4

Ta metoda jest identyczna z metodą WriteLong .

Składnia:

```
Call MQMessage.WriteInt4(value&)
```

Parametr *wartość &* Long. Wartość, która ma zostać zapisana.

Metoda WriteLong

Ta metoda przyjmuje podpisaną 4-bajtową liczbę całkowitą i zapisuje ją w buforze danych komunikatów jako 4-bajtowy numer binarny rozpoczynający się od bajtu przywołanego przez DataOffset. Zastępuje on wszystkie dane znajdujące się już na tych pozycjach w buforze i rozszerza długość buforu (MQMessage.MessageLength), jeśli jest to konieczne.

Wartość DataOffset jest zwiększana o 4, jeśli metoda zakończy się powodzeniem.

Metoda przekształca się w reprezentację binarną określoną przez właściwość MQMessage.Encoding .

Zdefiniowane w: Klasa MQMessage

Składnia:

```
Call MQMessage.WriteLong(value&)
```

Parametr *wartość &* Long. Wartość, która ma zostać zapisana.

Metoda WriteNullTerminatedString

Ta metoda wykonuje normalny łańcuch WriteString i dopełnia wszystkie pozostałe bajty aż do określonej długości z wartością NULL. Jeśli liczba bajtów zapisanych przez początkowy łańcuch zapisu jest równa określonej długości, wówczas nie są zapisywane żadne znaki puste. Jeśli liczba bajtów przekracza określoną długość, to jest ustawiony błąd (kod przyczyny MQRC_WRITE_VALUE_ERROR).

Wartość DataOffset jest zwiększana o określoną długość, jeśli metoda zakończy się powodzeniem.

Zdefiniowane w: Klasa MQMessage

Składnia:

```
Call MQMessage.WriteNullTerminatedString(value$, length&)
```

Parametry:
wartość $String. Wartość, która ma zostać zapisana.

długość i długość. Długość pola łańcucha w bajtach.

Metoda WriteShort

Ta metoda przyjmuje 2-bajtową liczbę całkowitą i zapisuje ją w buforze danych komunikatu jako dwubajtowy numer binarny rozpoczynający się w bajcie przywołanym przez DataOffset. Zastępuje on wszystkie dane znajdujące się na tych pozycjach w buforze, a w razie potrzeby rozszerzy długość buforu (MQMessage.MessageLength).

Wartość DataOffset jest zwiększana o 2, jeśli metoda powiedzie się.

Metoda przekształca się w reprezentację binarną określoną przez właściwość MQMessage.Encoding .

Zdefiniowane w: Klasa MQMessage

Składnia:

```
Call MQMessage.WriteShort(value%)
```

Parametr *wartość%* Liczba całkowita. Wartość, która ma zostać zapisana.

Metoda WriteString

Ta metoda pobiera łańcuch ActiveX i zapisuje go w buforze danych komunikatu, rozpoczynając od bajtu przywołanego przez element DataOffset. Zastępuje on wszystkie dane znajdujące się na tych pozycjach w buforze, a w razie potrzeby rozszerzy długość buforu (MQMessage.MessageLength).

DataOffset jest zwiększane o długość łańcucha w bajtach, jeśli metoda zakończy się powodzeniem.

Metoda przekształca znaki w stronę kodową określoną za pomocą właściwości MQMessage.CharacterSet .

Zdefiniowane w: Klasa MQMessage

Składnia:

```
Call MQMessage.WriteString(value$)
```

Parametr *wartość $* Łańcuch. Wartość, która ma zostać zapisana.

Metoda WriteUInt2

Ta metoda przyjmuje podpisaną 4-bajtową liczbę całkowitą i zapisuje ją w buforze danych komunikatu jako dwubajtowy, niepodpisany numer binarny, rozpoczynający się od bajtu, do którego odwołuje się DataOffset. Zastępuje on wszystkie dane znajdujące się już na tych pozycjach w buforze i rozszerza długość buforu (MQMessage.MessageLength), jeśli jest to konieczne.

Wartość DataOffset jest zwiększana o 2, jeśli metoda powiedzie się.

Metoda przekształca się w reprezentację binarną określoną przez właściwość MQMessage.Encoding . Podana wartość powinna miewać się w zakresie od 0 do 2 * *16-1. Jeśli nie jest to metoda, zwraca wartość CompletionCode MQCC_FAILED i ReasonCode MQRC_WRITE_VALUE_ERROR.

Zdefiniowane w: Klasa MQMessage

Składnia:

```
Call MQMessage.WriteUInt2(value&)
```

Parametr *wartość &* Long. Wartość, która ma zostać zapisana.

WriteUnsigned-metoda typu Byte

Ta metoda przyjmuje podpisaną 2-bajtową liczbę całkowitą i zapisuje ją w buforze danych komunikatów jako jednobajtowy niepodpisany numer binarny, rozpoczynający się od znaku, do którego odwołuje się DataOffset. Zastępuje on wszystkie dane znajdujące się już na tych pozycjach w buforze i rozszerza długość buforu (MQMessage.MessageLength), jeśli jest to konieczne.

Wartość DataOffset jest zwiększana o 1, jeśli metoda zakończy się powodzeniem.

Podana wartość powinna miewać się w zakresie od 0 do 255. Jeśli nie jest to metoda, zwraca wartość CompletionCode MQCC_FAILED i ReasonCode MQRC_WRITE_VALUE_ERROR.

Zdefiniowane w:
Klasa MQMessage

Składnia:

```
Call MQMessage.WriteUnsignedByte(value%)
```

Parametr *wartość%* Liczba całkowita. Wartość, która ma zostać zapisana.

Metoda WriteUTF

Ta metoda pobiera łańcuch ActiveX i zapisuje go w buforze danych komunikatów w bieżącej pozycji w formacie UTF. Zapis danych składa się z dwubajtowej długości, po której następują dane znakowe. DataOffset jest zwiększane o długość łańcucha, jeśli metoda zakończy się powodzeniem.

Zdefiniowane w:
Klasa MQMessage

Składnia: Wywołanie *MQMessage*. **WriteUTF** (wartość$)

parametr:
wartość $String. Wartość, która ma zostać zapisana.

Klasa opcji MQPutMessage

Ta klasa hermetykuje różne opcje, które sterują działaniem umieszczania komunikatu w kolejce produktu IBM MQ .

Zawieranie

Klasa opcji MQPutMessagejest zawarta w klasie MQSession.

Tworzenie

Opcja **Nowy** powoduje utworzenie nowego obiektu opcji MQPutMessagei ustawia wszystkie jego właściwości na wartości początkowe.

Alternatywnie można użyć metody AccessPutMessageOptions klasy MQSession.

Składnia

Dim *pmo* **Jako Nowy Opcje programu MQPutMessage** or

Ustaw *pmo* = **Nowy Opcje programu MQPutMessage**

Właściwości

- “Właściwość CompletionCode” na stronie 748.
- “Właściwość opcji” na stronie 748.
- “Właściwość ReasonCode” na stronie 748.
- “Właściwość ReasonName” na stronie 748.

- “Właściwość RecordFields” na stronie 749.
- “Właściwość ResolvedQueueManagerName” na stronie 749.
- “Właściwość Nazwa ResolvedQueue” na stronie 749.

Metody

- “ClearError-metoda kodów” na stronie 749.

Właściwość CompletionCode

Tylko do odczytu. Zwraca kod zakończenia ustawiony przez ostatnią metodę lub dostęp do właściwości wystawiony dla obiektu.

Zdefiniowane w: klasa opcji MQPutMessage

Typ danych: Long

Wartości:

- MQCC_OK
- MQCC_WARNING,
- MQCC_FAILED

Składnia: Do pobrania: *completioncode & = PutOpts* **.CompletionCode**

Właściwość opcji

Odczyt-zapis. Pole Opcje MQPMO. Wartością początkową tego pola jest MQPMO_NONE. Więcej informacji na ten temat zawiera sekcja Opcje MQPMO.

Zdefiniowana w: MQPutMessage-Klasa opcji.

Typ danych: Long

Składnia: Aby uzyskać: *options & = PutOpts* **.Opcje**

Aby ustawić: *PutOpts* **.Opcje** = *opcje &*

Opcje MQPMO_PASS_IDENTITY_CONTEXT i MQPMO_PASS_ALL_CONTEXT nie są obsługiwane.

Właściwość ReasonCode

Tylko do odczytu. Zwraca kod przyczyny ustawiony przez ostatnią metodę lub dostęp do właściwości wystawiony dla obiektu.

Zdefiniowane w: klasa opcji MQPutMessage

Typ danych: Long

Wartości:

- Patrz Kody zakończenia i przyczyny interfejsu API.

Składnia: Do pobrania: *kod_przyczyny & = PutOpts* **.ReasonCode**

Właściwość ReasonName

Tylko do odczytu. Zwraca nazwę symboliczną najnowszego kodu przyczyny. Na przykład: "MQRC_QMGR_NOT_AVAILABLE".

Zdefiniowane w: klasa opcji MQPutMessage

Typ danych: String

Wartości:

- Patrz Kody zakończenia i przyczyny interfejsu API.

Składnia: aby uzyskać: *reasonname $= PutOpts* **.ReasonName**

Właściwość RecordFields

Odczyt-zapis. Flagi wskazujące, które pola mają być dostosowywane w zależności od kolejki podczas umieszczania komunikatu na liście dystrybucyjnej. Wartością początkową jest zero.

Ta właściwość odpowiada flagom PutMsgRecFields w strukturze MQI MQPMO. W interfejsie MQI te flagi kontrolują, które pola (w strukturze MQPMR) są obecne i używane przez komendę MQPUT. W obiekcie opcji MQPutMessagete pola są zawsze obecne, a więc flagi mają wpływ tylko na to, które pola są używane przez komendę Put.

Zdefiniowane w:
Klasa opcji MQPutMessage

Typ danych:
Długa liczba całkowita

Składnia: Aby uzyskać: *recordfields & = PutOpts* **.RecordFields**

Aby ustawić: *PutOpts*. **RecordFields** = *recordfields &*

Właściwość ResolvedQueueManagerName

Tylko do odczytu. Pole nazwy ResolvedQMgrproduktu MQPMO. Szczegółowe informacje można znaleźć w sekcji ResolvedQMgrNazwa (MQCHAR48) . Wartością początkową jest wszystkie odstępy.

Zdefiniowane w: klasa opcji MQPutMessage

Typ danych: Łańcuch o długości 48 znaków

Składnia: aby uzyskać informacje: *qmgr $= PutOpts* **.ResolvedQueueManagerName**

Właściwość Nazwa ResolvedQueue

Tylko do odczytu. Pole MQPMO ResolvedQName . Szczegółowe informacje na ten temat zawiera sekcja ResolvedQName (MQCHAR48) . Wartością początkową jest wszystkie odstępy.

Zdefiniowane w: klasa opcji MQPutMessage

Typ danych: Łańcuch o długości 48 znaków

Składnia: aby uzyskać: *qname $= PutOpts* **.ResolvedQueueNazwa**

ClearError-metoda kodów

Resetuje parametr CompletionCode do wartości MQCC_OK i ReasonCode na wartość MQRC_NONE zarówno dla klasy opcji MQPutMessage, jak i klasy MQSession.

Zdefiniowane w:
Klasa opcji MQPutMessage

Składnia:
Call *PutOpts* **.ClearErrorCodes ()**

Klasa opcji MQGetMessage

Ta klasa hermetykuje różne opcje, które sterują działaniem pobierania komunikatu z kolejki produktu IBM MQ .

Zawieranie

Klasa opcji MQGetMessagejest zawarta w klasie MQSession.

Tworzenie

Opcja **Nowy** powoduje utworzenie nowego obiektu opcji MQGetMessagei ustawia wszystkie jego właściwości na wartości początkowe.

Alternatywnie można użyć metody AccessGetMessageOptions klasy MQSession.

Właściwości

- "Właściwość CompletionCode" na stronie 750
- "Właściwość MatchOptions" na stronie 750
- "Właściwość opcji" na stronie 751
- "Właściwość ReasonCode" na stronie 751
- "Właściwość ReasonName" na stronie 751
- "Właściwość Nazwa ResolvedQueue" na stronie 751
- "Właściwość WaitInterval" na stronie 751

Metody

- "ClearError-metoda kodów" na stronie 751

Składnia

Dim *gmo* **Jako nowe opcje MQGetMessage** lub

Ustaw *gmo* = **Nowe opcje MQGetMessage**

Właściwość CompletionCode

Tylko do odczytu. Zwraca kod zakończenia ustawiony przez ostatnią metodę lub dostęp do właściwości wystawiony dla obiektu.

Zdefiniowana w: MQGetMessage-Klasa opcji.

Typ danych: Long

Wartości:

- MQCC_OK
- MQCC_WARNING,
- MQCC_FAILED

Składnia: Aby uzyskać: *completioncode & = GetOpts* **.CompletionCode**

Właściwość MatchOptions

Odczyt-zapis. Opcje kontrolujące kryteria wyboru używane dla komendy MQGET. Wartością początkową jest MQMO_MATCH_MSG_ID + MQMO_MATCH_CORREL_ID.

Zdefiniowane w:
Klasa opcji MQGetMessage

Typ danych:
Długa liczba całkowita

Wartości:
Patrz MatchOptions (MQLONG).

Składnia: Aby uzyskać informacje: *matchoptions & = GetOpts*. **MatchOptions**

Aby ustawić: *GetOpts*. **MatchOptions** = *matchoptions &*

Właściwość opcji

Odczyt-zapis. Pole Opcje MQGMO. Szczegółowe informacje na ten temat zawiera sekcja Opcje . Wartością początkową jest MQGMO_NO_WAIT.

Zdefiniowana w: MQGetMessage-Klasa opcji.

Typ danych: Long

Składnia: Do pobrania: *opcje & = GetOpts* **.Opcje** Do ustawienia: *GetOpts* **.Opcje** = *opcje &*

Właściwość ReasonCode

Tylko do odczytu. Zwraca kod przyczyny ustawiony przez ostatnią metodę lub dostęp do właściwości wystawiony dla obiektu.

Zdefiniowane w: klasa opcji MQGetMessage

Typ danych: Long

Wartości:

- Patrz Kody zakończenia i przyczyny interfejsu API.

Składnia: do pobrania: *kod_przyczyny & = GetOpts* **.ReasonCode**

Właściwość ReasonName

Tylko do odczytu. Zwraca nazwę symboliczną najnowszego kodu przyczyny. Na przykład: "MQRC_QMGR_NOT_AVAILABLE". **Zdefiniowane w:** klasa opcji MQGetMessage

Typ danych: String

Wartości:

- Patrz Kody zakończenia i przyczyny interfejsu API.

Składnia: aby uzyskać następujące informacje: *reasonname $= MQGetMessageOptions* **.ReasonName**

Właściwość Nazwa ResolvedQueue

Tylko do odczytu. Pole MQGMO ResolvedQName . Szczegółowe informacje na ten temat zawiera sekcja ResolvedQName (MQCHAR48) . Wartością początkową jest wszystkie odstępy.

Zdefiniowane w: klasa opcji MQGetMessage

Typ danych: Łańcuch o długości 48 znaków

Składnia: aby uzyskać: *qname $= GetOpts* **.ResolvedQueueNazwa**

Właściwość WaitInterval

Odczyt/zapis. Pole MQGMO WaitInterval . Maksymalny czas (w milisekundach) oczekiwania na nadejście odpowiedniego komunikatu. W przypadku, gdy zażądano działania wait, właściwość Options. To pole ma wartość początkową równą 0. Szczegółowe informacje na temat opcji MQGMO można znaleźć w sekcji MQGMO.

Zdefiniowane w: klasa opcji MQGetMessage

Typ danych: Long

Składnia: Aby uzyskać: *wait & = GetOpts* **.WaitInterval**

Aby ustawić: *GetOpts* **.WaitInterval** = wait &

ClearError-metoda kodów

Resetuje parametr CompletionCode do wartości MQCC_OK i ReasonCode na wartość MQRC_NONE zarówno dla klasy opcji MQGetMessage, jak i klasy MQSession.

Zdefiniowane w:
Klasa opcji MQGetMessage

Składnia:
Call *GetOpts* **.ClearErrorCodes ()**

Klasa MQDistributionList

Ta klasa hermetyzuje kolekcję kolejek-lokalnych, zdalnych lub aliasów dla danych wyjściowych.

Tworzenie

new -tworzy nowy obiekt MQDistributionList .

Alternatywnie można użyć metody AddDistributionList klasy MQQueueManager .

Właściwości

- "Właściwość identyfikatora AlternateUser" na stronie 752
- "Właściwość CloseOptions" na stronie 752
- "Właściwość CompletionCode" na stronie 753
- "Właściwość ConnectionReference" na stronie 753
- "Właściwość FirstDistributionListItem" na stronie 753
- "Właściwość IsOpen" na stronie 753
- "Właściwość OpenOptions" na stronie 754
- "Właściwość ReasonCode" na stronie 754
- "Właściwość ReasonName" na stronie 754

Metoda

- "Metoda AddDistributionListItem" na stronie 754
- "ClearError-metoda kodów" na stronie 755
- "Metoda zamknięcia" na stronie 755
- "Metoda otwierania" na stronie 755
- "metoda PUT" na stronie 755

Składnia

Dim *distlist.* **A s New MQDistributionList** lub **Set** *distlist* = **New MQDistributionList**

Właściwość identyfikatora AlternateUser

Odczyt-zapis. Alternatywny identyfikator użytkownika używany do sprawdzania poprawności dostępu do listy kolejek po ich otwarciu.

Zdefiniowane w:
Klasa MQDistributionList

Typ danych:
Łańcuch o długości 12 znaków

Składnia: Aby uzyskać: *altuser $= MQDistributionList*. **IdentyfikatorAlternateUser**

Aby ustawić: *MQDistributionList*. **AlternateUserId** = *altuser $*

Właściwość CloseOptions

Odczyt-zapis. Opcje używane do sterowania tym, co się dzieje, gdy lista dystrybucyjna jest zamknięta. Wartością początkową jest MQCO_NONE.

Zdefiniowane w:
Klasa MQDistributionList

Typ danych:
Długa liczba całkowita

Wartości:

- MQCO_NONE
- MQCO_DELETE
- MQCO_DELETE_PURGE

Składnia: Aby uzyskać: *closeopt & = MQDistributionList*. **CloseOptions**

Aby ustawić: *MQDistributionList*. **CloseOptions** = *closeopt &*

Właściwość CompletionCode

Tylko do odczytu. Kod zakończenia ustawiony przez ostatnią metodę lub dostęp do właściwości wystawiony dla obiektu.

Zdefiniowane w:
Klasa MQDistributionList

Typ danych:
Długa liczba całkowita

Wartości:

- MQCC_OK
- MQCC_WARNING,
- MQCC_FAILED

Składnia: do pobrania: *completioncode & = MQDistributionList*. **CompletionCode**

Właściwość ConnectionReference

Odczyt-zapis. Menedżer kolejek, do którego należy lista dystrybucyjna.

Zdefiniowane w:
Klasa MQDistributionList

Typ danych:
MQQueueManager

Składnia: Aby uzyskać następujące informacje: *set queuemanager = MQDistributionList*. **ConnectionReference**

Aby ustawić: *set MQDistributionList*. **ConnectionReference** = *menedżer_kolejek*

Właściwość FirstDistributionListItem

Tylko do odczytu. Pierwszy obiekt pozycji listy dystrybucyjnej powiązany z listą dystrybucyjną.

Zdefiniowane w:
Klasa MQDistributionList

Typ danych:
Element MQDistributionList

Wartości:

Składnia: Aby uzyskać: *set distributionlistitem = MQDistributionList*. **FirstDistributionListItem**

Właściwość IsOpen

Tylko do odczytu.

Zdefiniowane w:
Klasa MQDistributionList

Typ danych:
wartość boolowska

Wartości:

- PRAWDA (-1)
- FALSE (0)

Składnia: Aby uzyskać: *IsOpen = MQDistributionList*. **IsOpen**

Właściwość OpenOptions

Odczyt-zapis. Opcje, które mają być używane, gdy lista dystrybucyjna jest otwarta.

Zdefiniowane w:
Klasa MQDistributionList

Typ danych:
Długa liczba całkowita

Wartości:
Więcej informacji zawiera sekcja Opcje MQPMO.

Składnia: Do pobrania: *openopt & = MQDistributionList*. **OpenOptions**

Aby ustawić wartość: *MQDistributionList.* **OpenOptions** = openopt &

Właściwość ReasonCode

Tylko do odczytu. Kod przyczyny ustawiony przez ostatnią metodę lub dostęp do właściwości wystawiony dla obiektu.

Zdefiniowane w:
Klasa MQDistributionList

Typ danych:
Długa liczba całkowita

Wartości:
Patrz Kody zakończenia i przyczyny interfejsu API.

Składnia: do pobrania: *kod_przyczyny & = MQDistributionList*. **ReasonCode**

Właściwość ReasonName

Tylko do odczytu. Nazwa symboliczna dla parametru ReasonCode. Na przykład "MQRC_QMGR_NOT_AVAILABLE".

Zdefiniowane w:
Klasa MQDistributionList

Typ danych:
Łańcuch

Wartości:
Patrz Kody zakończenia i przyczyny interfejsu API.

Składnia: aby uzyskać: *reasonname $= MQDistributionList*. **ReasonName**

Metoda AddDistributionListItem

Za pomocą tej metody można utworzyć nowy obiekt `MQDistributionListItem` i powiązać go z obiektem listy dystrybucyjnej. Parametr nazwy kolejki jest obowiązkowy.

Ta metoda wstawia nowy element listy dystrybucyjnej jako pierwszy element na istniejącej liście. W szczególności ta metoda tworzy następującą konfigurację:

- Na liście dystrybucyjnej ustawia ona właściwość **FirstDistributionListItem** , tak aby wskazywała na nową pozycję listy dystrybucyjnej.
- W nowej pozycji listy dystrybucyjnej ustawia ona następujące właściwości:
  - Ustawia ona właściwość **DistributionList** tak, aby wskazywała na listę dystrybucyjną.
  - Ustawia właściwość **PreviousDistributionListItem** na wartość NULL.
  - Ustawia ona właściwość **NextDistributionListItem** tak, aby wskazywał na pozycję listy dystrybucyjnej, która była wcześniej pierwsza, lub wartość null, jeśli na liście nie ma poprzednich elementów.

Nie można użyć tej metody w celu dodania nowej pozycji, gdy lista dystrybucyjna jest otwarta.

Zdefiniowane w:
Klasa MQDistributionList

Składnia: set distributionlistitem = *MQDistributionList* **.AddDistributionListItem** (QName$, QMgrName$)

Parametry:

QName$ Łańcuch. Nazwa kolejki produktu IBM MQ .

QMgrName$ Łańcuch. Nazwa menedżera kolejek produktu IBM MQ .

ClearError-metoda kodów

Resetuje parametr CompletionCode do wartości MQCC_OK i ReasonCode na wartość MQRC_NONE zarówno dla klasy MQDistributionList , jak i klasy MQSession.

Zdefiniowane w:
Klasa MQDistributionList

Składnia:

```
Call MQDistributionList.ClearErrorCodes()
```

Metoda zamknięcia

Zamyka listę dystrybucyjną przy użyciu bieżącej wartości opcji Zamknij.

Zdefiniowane w:
Klasa MQDistributionList

Składnia:
Call *MQDistributionList.* **Close** ()

Metoda otwierania

Otwiera każdą z kolejek określonych przez **QueueName** i (tam gdzie jest to stosowne) **QueueManagerName** właściwości pozycji listy dystrybucyjnej powiązanych z bieżącym obiektem, przy użyciu bieżącej wartości identyfikatora AlternateUser.

Zdefiniowane w:
Klasa MQDistributionList

Składnia:

```
Call MQDistributionList.Open()
```

metoda PUT

Umieszcza komunikat w każdej z kolejek identyfikowanych przez pozycje listy dystrybucyjnej powiązane z listą dystrybucyjną.

Zdefiniowane w:

Klasa MQDistributionList

Składnia

Wywołaj komendę MQDistributionList. **Umieść** (komunikat, opcje PutMsg&)

Parametry

Komunikat Obiekt MQMessage reprezentujący komunikat, który ma zostać umieszczony.

PutMsgOpcje MQPutMessageObiekt opcji zawierający opcje służące do sterowania operacją put. Jeśli nie zostanie podana, zostaną użyte domyślne opcje PutMessage.

Ta metoda przyjmuje obiekt MQMessage jako parametr. Następujące właściwości elementu listy dystrybucyjnej mogą zostać zmienione w wyniku użycia tej metody:

- CompletionCode
- ReasonCode
- ReasonName
- MessageId
- MessageIdHex
- CorrelationId
- CorrelationIdHex
- GroupId
- GroupIdHex
- Opinie
- AccountingToken
- AccountingTokenHex

MQDistributionList-klasa elementu

Ta klasa hermetykuje struktury MQOR, MQRR i MQPMR i wiąże je z listą dystrybucyjną będącą właścicielem.

Tworzenie

Należy użyć metody AddDistributionListItem klasy MQDistributionList .

Właściwości

Metody

- "Właściwość AccountingToken" na stronie 757.
- "Właściwość Hex AccountingToken" na stronie 758.
- "Właściwość CompletionCode" na stronie 758.
- "Właściwość CorrelationId" na stronie 758.
- "Właściwość CorrelationIdHex" na stronie 758.
- "Właściwość DistributionList" na stronie 759.
- "Właściwość sprzężenia zwrotnego" na stronie 759.
- "Właściwość GroupId" na stronie 759.
- "Właściwość GroupIdHex" na stronie 759.

- “Właściwość MessageId” na stronie 760.
- “Właściwość Hex MessageId” na stronie 760.
- “Właściwość NextDistributionListItem” na stronie 760.
- “Właściwość PreviousDistributionListItem” na stronie 760.
- “Właściwość Nazwa QueueManager” na stronie 760.
- “Właściwość QueueName” na stronie 761.
- “Właściwość ReasonCode” na stronie 761.
- “Właściwość ReasonName” na stronie 761.
- “ClearError-metoda kodów” na stronie 761.

Właściwości:

- Właściwość AccountingToken
- Właściwość Hex AccountingToken
- Właściwość CompletionCode
- Właściwość CorrelationId
- Właściwość CorrelationIdHex
- Właściwość DistributionList
- Właściwość sprzężenia zwrotnego
- Właściwość GroupId
- Właściwość GroupIdHex
- Właściwość MessageId
- Właściwość Hex MessageId
- Właściwość NextDistributionListItem
- Właściwość PreviousDistributionListItem
- Właściwość Nazwa QueueManager
- Właściwość QueueName
- Właściwość ReasonCode
- Właściwość ReasonName

Metody:

- ClearError-metoda kodów

Tworzenie:

Należy użyć metody AddDistributionListItem klasy MQDistributionList .

Właściwość AccountingToken

Odczyt-zapis. Element AccountingToken , który ma zostać umieszczony w kolejce komunikatów MQPMR komunikatu podczas umieszczania w kolejce. Jej wartością początkową jest wszystkie wartości NULL.

Zdefiniowane w:
MQDistributionList-klasa elementu

Typ danych:
Łańcuch zawierający 32 znaki

Składnia: Aby uzyskać: *accountingtoken $= MQDistributionListItem*. **AccountingToken**

Aby ustawić wartość: *MQDistributionListItem*. **AccountingToken** = *accountingtoken $*

Właściwość Hex AccountingToken

Odczyt-zapis. Element AccountingToken , który ma zostać umieszczony w kolejce komunikatów MQPMR komunikatu podczas umieszczania w kolejce.

Co dwa znaki w łańcuchu reprezentują szesnastkowy odpowiednik pojedynczego znaku ASCII. Na przykład para znaków "6" i "1" oznacza pojedynczy znak "A", a para znaków "6" i "2" reprezentuje pojedynczy znak "B" itd.

Należy podać 64 poprawne znaki szesnastkowe.

Jego początkowa wartość to "0 ... 0".

Zdefiniowane w:
MQDistributionList-klasa elementu

Typ danych:
Łańcuch o długości 64 znaków szesnastkowych, który zawiera 32 znaki ASCII.

Składnia: Aby uzyskać: *accountingtokenh = $= MQDistributionList*. **AccountingTokenHex**

Aby ustawić wartość: *MQDistributionListItem*. **AccountingTokenHex** = *accountingtokenh $*

Właściwość CompletionCode

Tylko do odczytu. Kod zakończenia ustawiony przez ostatnie otwarcie lub żądanie umieszczenia wydane dla obiektu listy dystrybucyjnej będącego właścicielem.

Zdefiniowane w:
MQDistributionList-klasa elementu

Typ danych:
Długa liczba całkowita

Wartości:

- MQCC_OK
- MQCC_WARNING,
- MQCC_FAILED

Składnia: Do pobrania: *kod_zakończenia $= MQDistributionList*. **CompletionCode**

Właściwość CorrelationId

Odczyt-zapis. Identyfikator CorrelId , który ma zostać włączony do rekordu MQPMR komunikatu podczas umieszczania w kolejce. Jej wartością początkową jest wszystkie wartości NULL.

Zdefiniowane w:
MQDistributionList-klasa elementu

Typ danych:
Łańcuch 24 znaków

Składnia: Aby uzyskać: *correlid $= MQDistributionListElement*. **CorrelationId**

Aby ustawić wartość: *MQDistributionListItem*. **CorrelationId** = *correlid $*

Właściwość CorrelationIdHex

Odczyt-zapis. Identyfikator CorrelId , który ma zostać włączony do rekordu MQPMR komunikatu podczas umieszczania w kolejce.

Co dwa znaki w łańcuchu reprezentują szesnastkowy odpowiednik pojedynczego znaku ASCII. Na przykład para znaków "6" i "1" oznacza pojedynczy znak "A", a para znaków "6" i "2" reprezentuje pojedynczy znak "B" itd.

Należy podać 48 poprawnych znaków szesnastkowych.

Jego początkowa wartość to "0 .. 0".

Zdefiniowane w:
MQDistributionList-klasa elementu

Typ danych:
Łańcuch 48 znaków szesnastkowych, reprezentujący 24 znaki ASCII.

Składnia: Aby uzyskać informacje: *correlidh $= MQDistributionList*. **CorrelationIdHex**

Aby ustawić wartość: *MQDistributionListItem*. **CorrelationIdHex** = *correlidh $*

Właściwość DistributionList

Tylko do odczytu. Lista dystrybucyjna, z którą powiązany jest ten element listy dystrybucyjnej.

Zdefiniowane w:
MQDistributionList-klasa elementu

Typ danych:
MQDistributionList

Składnia: Aby uzyskać: *set distributionlist = MQDistributionListItem*. **DistributionList**

Właściwość sprzężenia zwrotnego

Odczyt-zapis. Wartość sprzężenia zwrotnego, która ma być zawarta w komunikacie MQPMR komunikatu umieszczanego w kolejce.

Zdefiniowane w:
MQDistributionList-klasa elementu

Typ danych:
Długa liczba całkowita

Wartości:
Patrz Feedback (MQLONG).

Składnia: Aby uzyskać informacje: *feedback & = MQDistributionListItem*. **Opinie**

Aby ustawić wartość: *MQDistributionListItem*. **Opinia** = *opinia i*

Właściwość GroupId

Odczyt-zapis. Element GroupId , który ma zostać włączony do rekordu MQPMR komunikatu podczas umieszczania w kolejce. Jej wartością początkową jest wszystkie wartości NULL.

Zdefiniowane w:
MQDistributionList-klasa elementu

Typ danych:
Łańcuch 24 znaków

Składnia: Do pobrania: *groupid $= MQDistributionList*. **GroupId**

Aby ustawić wartość: *MQDistributionListItem*. **GroupId** = *groupid $*

Właściwość GroupIdHex

Odczyt-zapis. Element GroupId , który ma zostać włączony do rekordu MQPMR komunikatu podczas umieszczania w kolejce.

Co dwa znaki w łańcuchu reprezentują szesnastkowy odpowiednik pojedynczego znaku ASCII. Na przykład para znaków "6" i "1" oznacza pojedynczy znak "A", a para znaków "6" i "2" reprezentuje pojedynczy znak "B" itd.

Należy podać 48 poprawnych znaków szesnastkowych.

Jego początkowa wartość to "0 .. 0".

Zdefiniowane w:
MQDistributionList-klasa elementu

Typ danych:
Łańcuch o długości 48 znaków szesnastkowych, który zawiera 24 znaki ASCII.

Składnia: Do pobrania: *groupidh $= MQDistributionList*. **GroupIdHex**

Aby ustawić wartość: *MQDistributionListItem*. **GroupIdHex** = *groupidh $*

Właściwość MessageId

Odczyt-zapis. Element MessageId , który ma zostać włączony do rekordu MQPMR komunikatu podczas umieszczania w kolejce. Jej wartością początkową jest wszystkie wartości NULL.

Zdefiniowane w:
MQDistributionList-klasa elementu

Typ danych:
Łańcuch 24 znaków

Składnia: Do pobrania: *messageid $= MQDistributionList*. **MessageId**

Aby ustawić wartość: *MQDistributionListItem*. **MessageId** = *messageid $*

Właściwość Hex MessageId

Odczyt-zapis. Element MessageId , który ma zostać włączony do rekordu MQPMR komunikatu podczas umieszczania w kolejce.

Co dwa znaki w łańcuchu reprezentują szesnastkowy odpowiednik pojedynczego znaku ASCII. Na przykład para znaków "6" i "1" oznacza pojedynczy znak "A", a para znaków "6" i "2" reprezentuje pojedynczy znak "B" itd.

Należy podać 48 poprawnych znaków szesnastkowych.

Jego początkowa wartość to "0 .. 0".

Zdefiniowane w:
MQDistributionList-klasa elementu

Typ danych:
Łańcuch 48 znaków szesnastkowych, reprezentujący 24 znaki ASCII.

Składnia: Aby uzyskać: *messageidh $= MQDistributionListElement*. **MessageIdHex**

Aby ustawić wartość: *MQDistributionListItem*. **MessageIdHex** = *messageidh $*

Właściwość NextDistributionListItem

Tylko do odczytu. Następny obiekt pozycji listy dystrybucyjnej powiązany z tą samą listą dystrybucyjną.

Zdefiniowane w:
MQDistributionList-klasa elementu

Typ danych:
Element MQDistributionList

Składnia: Do pobrania: *set distributionlistitem = MQDistributionList.* **NextDistributionListItem**

Właściwość PreviousDistributionListItem

Tylko do odczytu. Poprzedni obiekt pozycji listy dystrybucyjnej powiązany z tą samą listą dystrybucyjną.

Zdefiniowane w:
MQDistributionList-klasa elementu

Typ danych:
Element MQDistributionList

Składnia: Aby uzyskać: *set distributionlistitem = MQDistributionListItem*. **PreviousDistributionListItem**

Właściwość Nazwa QueueManager

Odczyt-zapis. Nazwa menedżera kolejek produktu IBM MQ .

Zdefiniowane w:
MQDistributionList-klasa elementu

Typ danych:
Łańcuch o długości 48 znaków.

Składnia: aby uzyskać: *qmname $= MQDistributionListElement*. **QueueManagerName**

Aby ustawić wartość: *MQDistributionListItem*. **QueueManagerNazwa** = *qmname $*

Właściwość QueueName

Odczyt-zapis. Nazwa kolejki produktu IBM MQ .

Zdefiniowane w:
MQDistributionList-klasa elementu

Typ danych:
Łańcuch o długości 48 znaków.

Składnia: aby uzyskać: *qname $= MQDistributionListItem*. **QueueName**

Aby ustawić wartość: *MQDistributionListItem*. **QueueName** = *qname $*

Właściwość ReasonCode

Tylko do odczytu. Kod przyczyny ustawiony jako ostatni otwarty lub wysłany do obiektu listy dystrybucyjnej będącego właścicielem.

Zdefiniowane w:
MQDistributionList-klasa elementu

Typ danych:
Długa liczba całkowita

Wartości:
Patrz Kody zakończenia i przyczyny interfejsu API.

Składnia: Aby uzyskać:

```
reasoncode& = MQDistributionListItem.ReasonCode
```

Właściwość ReasonName

Tylko do odczytu. Nazwa symboliczna dla parametru ReasonCode. Na przykład "MQRC_QMGR_NOT_AVAILABLE".

Zdefiniowane w:
MQDistributionList-klasa elementu

Typ danych:
Łańcuch

Wartości:
Patrz Kody zakończenia i przyczyny interfejsu API.

Składnia: aby uzyskać: *reasonname $= MQDistributionListItem*. **ReasonName**

ClearError-metoda kodów

Resetuje parametr CompletionCode do wartości MQCC_OK i ReasonCode na wartość MQRC_NONE zarówno dla klasy elementu MQDistributionList, jak i klasy MQSession.

Zdefiniowane w:
MQDistributionList-klasa elementu

Składnia:

```
Call MQDistributionListItem.ClearErrorCodes
```

Śledzenie klas automatyzacji produktu IBM MQ dla elementu ActiveX

Informacje na temat narzędzia śledzenia udostępnionego dla klas automatyzacji produktu IBM MQ dla elementów ActiveX, wspólnych pułapek i pomocy na temat sposobów ich unikania.

W produkcie IBM MQ 9.0obsługa produktu IBM MQ dla produktu Microsoft Active X jest nieaktualna. Zalecaną technologią zastępczą są klasy IBM MQ dla .NET. Więcej informacji na ten temat zawiera sekcja Tworzenie aplikacji .NET.

W tej sekcji opisano dostępne narzędzia śledzenia oraz szczegółowe informacje na temat pułapek, które ułatwiają ich uniknięcie:

- “Sterowanie śledzeniem dla klas automatyzacji produktu IBM MQ dla elementu ActiveX” na stronie 762
- “Jeśli nie powiedzie się próba wykonania skryptu IBM MQ Automation Classes for ActiveX” na stronie 763
- “Kody przyczyny IBM MQ Klasy automatyzacji dla ActiveX” na stronie 763
- “Narzędzie na poziomie kodu” na stronie 766

Sterowanie śledzeniem dla klas automatyzacji produktu IBM MQ dla elementu ActiveX

Klasy automatyzacji produktu IBM MQ dla elementu ActiveX (MQAX) zawierają narzędzie do śledzenia, które ułatwia organizacji usług identyfikowanie tego, co się dzieje, gdy występuje problem.

Zawiera on ścieżki wykonywane po uruchomieniu skryptu MQAX . Jeśli nie masz problemu, uruchom śledzenie wyłączone, aby uniknąć niepotrzebnego korzystania z zasobów systemowych.

Dostępne są trzy zmienne środowiskowe, które można ustawić w celu sterowania procesem śledzenia:

- ŚLEDZENIE OMQ_TRACE
- OMQ_TRACE_PATH
- OMQ_TRACE_LEVEL

Uwaga: Określenie wartości *dowolna* dla zmiennej **OMQ_TRACE** powoduje przełączenie narzędzia śledzenia. Nawet jeśli zmienna **OMQ_TRACE** zostanie ustawiona na 0FF, śledzenie będzie nadal aktywne. Aby wyłączyć śledzenie, nie należy określać wartości parametru **OMQ_TRACE**.

1. Kliknij opcję **Uruchom** .
2. Kliknij opcję **Panel sterowania** .
3. Dwukrotnie kliknij **System**
4. Kliknij opcję **Zaawansowane** .
5. Kliknij opcję **Środowisko** .
6. W sekcji zatytułowanej **User variables for (username)**kliknij opcję **New** .
7. Wprowadź nazwę zmiennej i poprawną wartość w odpowiednich polach, a następnie kliknij przycisk **OK** .
8. Kliknij przycisk **OK** , aby zamknąć okno Zmienne środowiskowe.
9. Kliknij przycisk **OK** , aby zamknąć okno Właściwości systemu.
10. Zamknij okno Panel sterowania

Podejmując decyzję o miejscu, w którym mają być zapisywane pliki śledzenia, należy upewnić się, że użytkownik posiada wystarczające uprawnienia do zapisu oraz odczytu z dysku.

Włączenie śledzenia powoduje spowolnienie działania programu MQAX, ale nie wpływa to na wydajność środowisk ActiveX lub IBM MQ . Jeśli plik śledzenia nie jest już potrzebny, można go usunąć.

Nie można zmienić statusu zmiennej OMQ_TRACE podczas działania MQAX.

Nazwa i katalog pliku śledzenia

Nazwa pliku śledzenia ma postać `OMQ`*nnnnn*`.trc`, gdzie *nnnnn* jest identyfikatorem procesu ActiveX uruchomionego w danym momencie.

Tabela 103. Komendy i ich efekty

Komenda	Efekt
SET OMQ_TRACE_PATH = drive: \katalog	Ustawia katalog śledzenia, w którym zapisywany jest plik śledzenia.
USTAW ZMIENNĄ OMQ_TRACE_PATH =	Usuwa wszystkie istniejące ustawienia dla katalogu śledzenia. Gdy katalog śledzenia nie jest ustawiony, używany jest bieżący katalog roboczy (gdy uruchamiany jest element ActiveX).
ECHO %OMQ_TRACE_PATH%	Wyświetla bieżące ustawienie dla katalogu śledzenia w systemie Windows.
SET OMQ_TRACE = xxxxxxxx	Włącza śledzenie. Śledzenie można włączyć, umieszczając jeden lub więcej znaków po znaku '='. Na przykład: `SET OMQ_TRACE=yes` i `SET OMQ_TRACE=no`. W obu tych przykładach śledzenie jest włączone. To ustawienie obowiązuje tylko w przypadku pojedynczego okna/sesji.
SET OMQ_TRACE=	Wyłącza śledzenie.
ECHO %OMQ_TRACE%	Wyświetla zawartość zmiennej środowiskowej w systemie Windows.
SET	Wyświetla zawartość wszystkich zmiennych środowiskowych w systemie Windows.
USTAW WARTOŚĆ OMQ_TRACE_LEVEL = 9	Ustawia poziom śledzenia na 9. Wartości większe niż 9 nie generują żadnych dodatkowych informacji w pliku śledzenia.

Jeśli nie powiedzie się próba wykonania skryptu IBM MQ Automation Classes for ActiveX

Jeśli działanie skryptu IBM MQ Automation Classes for ActiveX nie powiedzie się, istnieje wiele źródeł informacji, które można przejrzeć.

Raport objawów pierwszego niepowodzenia

Niezależnie od narzędzia śledzenia, w przypadku nieoczekiwanych i wewnętrznych błędów, może zostać wygenerowany raport objawów pierwszego niepowodzenia.

Ten raport znajduje się w pliku o nazwie `OMQ`*nnnnn*`.fdc`, gdzie *nnnnn* jest numerem procesu ActiveX , który jest uruchomiony w danym momencie. Plik ten znajduje się w katalogu roboczym, z którego został uruchomiony element ActiveX lub w ścieżce określonej w zmiennej środowiskowej OMQ_PATH.

Inne źródła informacji

Produkt IBM MQ udostępnia różne dzienniki błędów i informacje o śledzeniu, w zależności od używanej platformy. Zapoznaj się z dziennikiem zdarzeń aplikacji Windows .

Kody przyczyny IBM MQ Klasy automatyzacji dla ActiveX

Kody przyczyn dla klas automatyzacji produktu IBM MQ dla ActiveX (MQAX), które mogą wystąpić oprócz kodów przyczyny MQI produktu IBM MQ .

Oprócz udokumentowanych dla interfejsu MQI produktu IBM MQ mogą wystąpić następujące kody przyczyny. Informacje o innych kodach można znaleźć w dzienniku zdarzeń aplikacji IBM MQ .

Tabela 104. Kody przyczyny i to, co oznaczają

Kod przyczyny	Wyjaśnienie
MQRC_LIBRARY_LOAD_ERROR (6000)	Nie można załadować jednej lub większej liczby bibliotek produktu IBM MQ . Sprawdź, czy wszystkie biblioteki produktu IBM MQ znajdują się w poprawnej ścieżce wyszukiwania w systemie, z którego korzysta użytkownik. Na przykład należy upewnić się, że katalogi zawierające biblioteki produktu IBM MQ znajdują się w katalogu PATH.
MQRC_CLASS_LIBRARY_ERROR (6001)	Jeden z wywołań IBM MQ `classlibrary` zwrócił nieoczekiwaną wartość **ReasonCode** lub **CompletionCode** . Szczegółowe informacje można znaleźć w raporcie objawów pierwszego niepowodzenia. Zanotuj ostatnio używaną metodę/właściwość i klasę, a następnie poinformuj IBM Wsparcie problemu.
MQRC_STRING_LENGTH_TOO_BIG (6002)	Podjęto próbę zapisu łańcucha formatu UTF o długości większej niż 65,535 bajtów do buforu komunikatów.
MQRC_WRITE_VALUE_ERROR (6003)	Używana jest wartość spoza zakresu, na przykład `msg.WriteByte (240)`.
MQRC_PACKED_DECIMAL_ERROR (6004)	Podjęto próbę odczytu upakowanej liczby dziesiętnej z buforu komunikatów, ale dane znajdujące się na wskaźniku danych nie są w poprawnym formacie spakowanego danych.
MQRC_FLOAT_CONVERSION_ERROR (6005)	Próbowano odczytać pojedynczy lub podwójny numer zmiennopozycyjny z buforu komunikatów, ale dane w wskaźniku danych nie są w odpowiednim formacie zmiennopozycyjnym.
MQRC_REOPEN_EXCL_INPUT_ERROR (6100)	Obiekt otwarty nie ma poprawnych ustawień **OpenOptions** i wymaga co najmniej jednej dodatkowej opcji. Wymagane jest niejawne ponowne otwarcie, ale uniemożliwiono zamknięcie, ponieważ kolejka jest otwarta do wyłącznego wejścia, a zamknięcie może spowodować, że inne osoby potencjalnie mogą uzyskać dostęp do kolejki. Ustaw wartości **OpenOptions** w sposób jawny, tak aby obejmował wszystkie ewentualności, tak aby niejawne ponowne otwarcie nie było wymagane.
MQRC_REOPEN_INQUIRE_ERROR (6101)	Obiekt otwarty nie ma poprawnych ustawień **OpenOptions** i wymaga co najmniej jednej dodatkowej opcji. Niejawne ponowne otwarcie jest wymagane, ale uniemożliwiono zamknięcie, ponieważ co najmniej jedna charakterystyka obiektu musi zostać sprawdzona dynamicznie przed zamknięciem, a wartości **OpenOptions** nie zawierają już opcji **MQOO_INQUIRE** . Ustaw wartości **OpenOptions** jawnie, tak aby uwzględniała opcję **MQOO_INQUIRE** .
MQRC_REOPEN_SAVED_CONTEXT_ERR (6102)	Obiekt otwarty nie ma poprawnych ustawień **OpenOptions** i wymaga co najmniej jednej dodatkowej opcji. Wymagane jest niejawne ponowne otwarcie, ale uniemożliwiono zamknięcie, ponieważ kolejka jest otwarta za pomocą opcji **MQOO_SAVE_ALL_CONTEXT** , a destrukcyjne wywołanie `Get` zostało wykonane wcześniej. Spowodowało to, że zachowana informacja o stanie została powiązana z otwartą kolejką, a informacje te zostaną zniszczone przez zamknięcie. Ustaw wartości **OpenOptions** w sposób jawny, tak aby obejmował wszystkie ewentualności, tak aby niejawne ponowne otwarcie nie było wymagane.
MQRC_REOPEN_TEMPORARY_Q_ERROR (6103)	Obiekt otwarty nie ma poprawnych ustawień **OpenOptions** i wymaga co najmniej jednej dodatkowej opcji. Wymagane jest niejawne ponowne otwarcie, ale uniemożliwiono zamknięcie, ponieważ kolejka jest kolejką lokalną definicji typu **MQQDT_TEMPORARY_DYNAMIC**, która zostałaby zniszczona przez zamknięcie. Ustaw wartości **OpenOptions** w sposób jawny, tak aby obejmował wszystkie ewentualności, tak aby niejawne ponowne otwarcie nie było wymagane.
MQRC_ATTRIBUTE_LOCKED (6104)	Podjęto próbę zmiany wartości lub atrybutu obiektu, gdy obiekt jest otwarty. Niektóre atrybuty, takie jak **AlternateUserId**, nie mogą być zmieniane, gdy obiekt jest otwarty.

Tabela 104. Kody przyczyny i to, co oznaczają (kontynuacja)

Kod przyczyny	Wyjaśnienie
MQRC_CURSOR_NOT_VALID (6105)	Kursor przeglądania dla otwartej kolejki został unieważniony, ponieważ był ostatnio używany przez niejawne ponowne otwarcie. Ustaw wartości **OpenOptions** w sposób jawny, tak aby obejmował wszystkie ewentualności, tak aby niejawne ponowne otwarcie nie było wymagane.
MQRC_ENCODING_ERROR (6106)	Kodowanie następnego elementu komunikatu musi mieć wartość kodowania **MQENC_NATIVE** do odczytu.
BŁĄD MQRC_STRUCID_ERROR (6107)	Struktura identyfikatora następnego elementu komunikatu, która pochodzi z 4 znaków zaczynając od wskaźnika danych, jest brakująca lub jest niespójna z typem zmiennej, do której element jest odczytywaany.
MQRC_NULL_POINTER (6108)	Podano pusty wskaźnik, w którym wymagany lub domniemany wskaźnik niezerowy jest wymagany. Przyczyną może być użycie jawnych deklaracji dla obiektów produktu IBM MQ , które są używane w języku Visual Basic lub Excel jako parametry do wywołań. Na przykład: • Z poziomu Visual Basic program `dim msg as Object` działa poprawnie, podczas gdy produkt `dim msg as MqMessage` może nie działać poprawnie. • Z poziomu Visual Basic, z określoną kolejką i ustawionym zestawem, program `dim msg as MqMessageq.put msg` działa poprawnie, podczas gdy z programu Excel ta komenda generuje wyjątek `MQRC_NULL_POINTER` .
MQRC_NO_CONNECTION_REFERENCE (6109)	Obiekt `MQQueue` utracił połączenie z obiektem `MQQueueManager` . Ta opcja zostanie wykonana, jeśli menedżer kolejek zostanie odłączony. Usuń obiekt `MQQueue` .
MQRC_NO_BUFFER (6110)	Bufor nie jest dostępny. W przypadku obiektu `MQMessage` nie można przydzielić buforu, ponieważ w stanie obiektu istnieje wewnętrzna niespójność.
MQRC_BINARY_DATA_LENGTH_ERROR (6111)	Długość danych binarnych jest niespójna z długością atrybutu docelowego. Wartość zero jest poprawną długością dla wszystkich atrybutów. 24 to poprawna długość dla atrybutu **CorrelationId** i atrybutu **MessageId** . Wartość 32 jest poprawną długością atrybutu **AccountingToken** .
MQRC_BUFFER_NOT_AUTOMATIC (6112)	Nie można zmienić wielkości buforu zdefiniowanego przez użytkownika i zarządzanego przez użytkownika. Ponieważ bufory komunikatów są zarządzane przez system, wskazuje to na wewnętrzną niespójność.
MQRC_INSUFFICIENT_BUFFER (6113)	Brak wystarczającej ilości miejsca w buforze po umieszczonym wskaźniku danych w celu dostosowania go do żądania. Może to być spowodowane tym, że bufor nie może być rezygnowany.
MQRC_INSUFFICIENT_DATA (6114)	Po umieszczonym wskaźniku danych nie ma wystarczających danych, aby zmieścić się w żądaniu odczytu. Zmniejsz bufor do odpowiedniej wielkości i ponownie odczytaj dane.
MQRC_DATA_OBCIĘTY (6115)	Dane zostały obcięte podczas kopiowania z jednego buforu do innego. Może to być spowodowane tym, że nie można zmienić wielkości buforu docelowego, lub ponieważ wystąpił problem z adresowaniem jednego lub innego buforu, lub dlatego, że bufor jest malejący z mniejszym wymianą.
MQRC_ZERO_LENGTH (6116)	Podano zerową długość, w przypadku której wymagana lub domniemana długość dodatnia jest wymagana.
MQRC_NEGATIVE_LENGTH (6117)	Podano ujemną długość, w której wymagana jest długość zerowa lub dodatnia.
MQRC_NEGATIVE_OFFSET (6118)	Podano ujemne przesunięcie w przypadku, gdy wymagane jest przesunięcie zerowe lub dodatnie.
MQRC_INCONSISTENT_FORMAT (6119)	Format następnego elementu komunikatu jest niespójny z typem zmiennej, do której element jest odczytyowany.

Tabela 104. Kody przyczyny i to, co oznaczają (kontynuacja)

Kod przyczyny	Wyjaśnienie
MQRC_INCONSISTENT_OBJECT_STATE (6120)	Między tym obiektem jest niespójność, która jest otwarta, a przywoływany obiekt `MQQueueManager` , który nie jest połączony.
MQRC_CONTEXT_OBJECT_NOT_VALID (6121)	Odwołanie do kontekstu `MQPutMessageOptions` nie odwołuje się do poprawnego obiektu `MQQueue` . Obiekt został wcześniej zniszczony.
MQRC_CONTEXT_OPEN_ERROR (6122)	Odwołanie do kontekstu `MQPutMessageOptions` odwołuje się do obiektu `MQQueue` , którego nie można otworzyć w celu ustanowienia kontekstu. Może to być spowodowane tym, że obiekt `MQQueue` ma nieodpowiednie opcje otwarcia. Sprawdź przywoływany kod przyczyny obiektu, aby ustalić przyczynę.
Błąd MQRC_STRUC_LENGTH_ERROR (6123)	Długość wewnętrznej struktury danych jest niespójna z jej treścią. W przypadku nagłówka `MQRMH` długość nie jest wystarczająca, aby pomieścić pola stałe i wszystkie dane przesunięcia.
MQRC_NOT_CONNECTED (6124)	Metoda nie powiodła się, ponieważ wymagane połączenie z menedżerem kolejek nie jest dostępne, a połączenie nie może zostać nawiązane niejawnie.
MQRC_NOT_OPEN (6125)	Metoda nie powiodła się, ponieważ obiekt IBM MQ nie jest otwarty, a otwarcie nie może być wykonane niejawnie.
MQRC_DISTRIBUTION_LIST_EMPTY (6126)	Próba otwarcia `MQDistributionList` nie powiodła się, ponieważ na liście dystrybucyjnej nie ma obiektów `MQDistributionListItem` . Czynność naprawczy: należy dodać co najmniej jeden obiekt `MQDistributionListItem` do listy dystrybucyjnej.
MQRC_INCONSISTENT_OPEN_OPTIONS (6127)	Metoda nie powiodła się, ponieważ obiekt jest otwarty, a opcje otwarcia są niespójne z wymaganą operacją. Czynność naprawczy: otwórz obiekt z odpowiednimi opcjami otwarcia, a następnie spróbuj ponownie.
MQRC_WRONG_VERSION (6128)	Metoda nie powiodła się, ponieważ podany lub napotkany numer wersji jest niepoprawny lub nie jest obsługiwany.

Narzędzie na poziomie kodu

Zespół serwisu IBM może zapytać, na jakim poziomie kodu został zainstalowany.

Aby to sprawdzić, uruchom program narzędziowy MQAXLEV.

W wierszu komend przejdź do katalogu zawierającego plik MQAX200.dll lub dodaj pełną długość ścieżki i wpisz:

```
MQAXLev MQAX200.dll > MQAXLEV.OUT
```

gdzie MQAXLEV.OUT jest nazwą pliku wyjściowego.

Jeśli zbiór wyjściowy nie zostanie określony, na ekranie zostaną wyświetlone szczegóły.

Przykładowy plik wyjściowy z narzędzia na poziomie kodu został szczegółowo opisany w poniższym przykładzie:

Przykładowy plik wyjściowy z narzędzia na poziomie kodu

```
5639-B43 (C) Copyright IBM Corp. 1996, 2024. ALL RIGHTS RESERVED.
     ***** Code Level is 5.1 *****
lib/mqole/mqole.cpp, mqole, p000, p000 L981119       1.8 98/08/21
lib/mqlsx/gmqdyn0a.c, mqlsx, p000, p000 L990212      1.6 99/02/11 16:40:24
lib/mqlsx/pc/gmqdyn1p.c, mqlsx, p000, p000 L990212   1.6 99/02/11 16:44:14
lib/mqlsx/xmqcsa.c, mqole, p000, p000 L990216        1.3 99/02/15 13:24:34
lib/mqlsx/xmqfdca.c, mqlsx, p000, p000 L990212       1.3 99/02/11 16:40:35
lib/mqlsx/xmqtrca.c, mqlsx, p000, p000 L990212       1.5 99/02/11 16:12:02
lib/mqlsx/xmqutila.c, mqlsx, p000, p000 L990212      1.3 99/02/11 16:40:40
lib/mqlsx/xmqutl1a.c, mqlsx, p000, p000 L990212      1.4 99/02/11 16:40:30
lib/mqlsx/xmqcnv1a.c, mqlsx, p000, p000 L990212      1.9 99/02/11 16:40:56
lib/mqlsx/xmqmsg.c, mqole, p000, p000 L990219        1.11 99/02/18 12:12:59
```

Interfejs ActiveX do interfejsu MQAI

Krótki przegląd interfejsów COM i ich zastosowanie w interfejsie MQAI zawiera sekcja “Korzystanie z interfejsu modelu obiektu komponentu (klasy automatyzacji produktuIBM MQ dla elementu ActiveX)” na stronie 686.

Interfejs MQAI umożliwia aplikacjom budowanie i wysyłanie komend PCF (Programmable Command Format) bez bezpośredniego uzyskiwania i formatowania buforów o zmiennej długości wymaganych dla PCF. Więcej informacji na temat interfejsu MQAI znajduje się w sekcji Interfejs administracyjny produktu IBM MQ (MQAI). Klasa MQAI ActiveX MQBag obudowuje worki danych obsługiwane przez interfejs MQAI w sposób, który może być używany w dowolnym języku, który obsługuje tworzenie obiektów COM, na przykład Visual Basic, C + +, Javai innych klientów skryptowych ActiveX .

Interfejs MQAI ActiveX jest przeznaczony do użytku z klasami MQAX udostępniających interfejs COM do interfejsu MQI. Więcej informacji na temat klas MQAX zawiera sekcja “Projektowanie aplikacji MQAX, które uzyskują dostęp do aplikacji innych niżActiveX” na stronie 687.

Interfejs ActiveX udostępnia pojedynczą klasę o nazwie MQBag. Ta klasa jest używana do tworzenia worków danych MQAI, a jej właściwości i metody są używane do tworzenia elementów danych i pracy z elementami danych w każdej z nich. Metoda Execute MQBag Execute wysyła dane o torbie do menedżera kolejek produktu IBM MQ jako komunikat PCF i zbiera odpowiedzi.

Więcej informacji na temat klasy MQBag, jej właściwości i metod zawiera sekcja “Klasa MQBag” na stronie 767.

Komunikat PCF jest wysyłany do określonego obiektu menedżera kolejek, opcjonalnie przy użyciu określonych kolejek żądań i odpowiedzi. Odpowiedzi są zwracane w nowym obiekcie MQBag. Pełny zestaw komend i odpowiedzi jest opisany w sekcji Definicje formatów komend programowalnych. Komendy mogą być wysyłane do dowolnego menedżera kolejek w sieci IBM MQ , wybierając odpowiednie kolejki żądań i odpowiedzi.

Klasa MQBag

Klasa, MQBag, jest używana do tworzenia obiektów MQBag zgodnie z wymaganiami. Po utworzeniu instancji klasa MQBag zwraca nowe odwołanie do obiektu MQBag.

Utwórz obiekt MQBag w języku Visual Basic w następujący sposób:

```
Dim mqbag As MQBag
Set mqbag = New MQBag
```

Właściwość MQBag

Właściwości obiektów MQBag zostały wyjaśnione na następującej liście:

- “Właściwość elementu” na stronie 768.

- “Właściwość liczebności” na stronie 769.
- “Właściwość opcji” na stronie 770.

Metody MQBag

Metody obiektów MQBag są wyjaśnione na następującej liście:

- “Dodaj metodę” na stronie 770.
- “Metoda AddInquiry” na stronie 771.
- “Wyczyść metodę” na stronie 771.
- “Metoda execute” na stronie 772.
- “Metoda FromMessage” na stronie 772.
- “Metoda ItemType” na stronie 773.
- “metoda usuwania” na stronie 773.
- “Metoda selektora” na stronie 774.
- “Metoda ToMessage” na stronie 775.
- “Metoda obcinania” na stronie 775.

Obsługa błędów

Jeśli podczas operacji na obiekcie MQBag wykryty zostanie błąd, w tym te, które zostały zwrócone do tego obiektu przez bazowy obiekt MQAX lub MQAI, zgłaszany jest wyjątek błędu. Klasa MQBag obsługuje interfejs COM ISupportErrorInfo, więc następujące informacje są dostępne dla procedury obsługi błędów:

- Numer błędu: składający się z kodu przyczyny IBM MQ dla wykrytego błędu i kodu narzędzia COM. Pole obiektu, zgodnie ze standardem COM, wskazuje obszar odpowiedzialności za błąd. W przypadku błędów wykrytych przez produkt IBM MQ zawsze jest to FACILITY_ITF.
- Źródło błędu: identyfikuje typ i wersję obiektu, który wykrył błąd. W przypadku błędów wykrytych podczas operacji MQBag źródłem błędu jest zawsze MQBag.MQBag1.
- Opis błędu: łańcuch zawierający nazwę symboliczną dla kodu przyczyny produktu IBM MQ .

Sposób uzyskiwania dostępu do informacji o błędzie zależy od języka skryptowego. Na przykład w języku Visual Basic informacje są zwracane w obiekcie Err, a kod przyczyny IBM MQ jest uzyskiwany przez odjęcie stałego błędu vbObjectz poziomu Err.Number.

ReasonCode = Err.Number -Błąd vbObject

Jeśli metoda Execute MQBag Execute wysyła komunikat PCF i zostanie odebrana odpowiedź, operacja zostanie uznana za pomyślną, mimo że wysłana komenda mogła się nie powiodła. W takim przypadku sam worek odpowiedzi zawiera kody przyczyny zakończenia i przyczyny błędu, zgodnie z opisem w sekcji Definicje formatów komend programowalnych.

Właściwość elementu

Przeznaczenie

Właściwość Item reprezentuje element w torbie. Jest on używany do ustawiania lub sprawdzania wartości elementu. Użycie tej właściwości odpowiada następującym wywoławczym MQAI:

- "ŁańcuchmqSet"
- "mqSetLiczba całkowita"
- "mqInquireLiczba całkowita"
- "ŁańcuchmqInquire"
- "TorbamqInquire"

w Skorowidz formatów komend programowalnych.

Format

Element (Selector, ItemIndex, Value)

Parametry

Selektor (VARIANT)-wejście
Selektor elementu, który ma zostać ustawiony lub wypytany.

Po zapytaniu o element jest to wartość domyślna MQSEL_ANY_USER_SELECTOR. Podczas ustawiania elementu domyślnie jest używana lista MQIA_LIST lub MQCA_LIST.

Jeśli wartość `Selector` nie jest typu long, wyniki MQRC_SELECTOR_TYPE_ERROR.

Ten parametr jest opcjonalny.

ItemIndex (LONG)-dane wejściowe
Ta wartość identyfikuje wystąpienie elementu określonego selektora, który ma zostać ustawiony lub zrzekłany. Wartość MQIND_NONE jest wartością domyślną.

Ten parametr jest opcjonalny.

Value (VARIANT)-wejście/wyjście
Zwracana wartość lub wartość, która ma zostać ustawiona. Po zapytaniu o element wartość zwracana może być typu long, string lub MQBag. Jednak podczas ustawiania elementu wartość musi być typu long lub string (jeśli nie), MQRC_ITEM_VALUE_ERROR powoduje błąd.

Wizualne wywołanie języka podstawowego

Podczas uzyskiwania informacji o wartości elementu w obrębie torby:

```
Value = mqbag[.Item]([Selector],
[ItemIndex])
```

Dla odwołań do MQBag:

```
Set abag = mqbag[.Item]([Selector].
[ItemIndex])
```

Aby ustawić wartość elementu w worku:

```
mqbag[.Item]([Selector],
[ItemIndex]) = Value
```

Właściwość liczebności

Przeznaczenie

Właściwość Liczba reprezentuje liczbę elementów danych w torbie. Ta właściwość odpowiada wywołaniu MQAI, "mqCountElementów" w Skorowidz formatów komend programowalnych.

Format

Liczba (*`Selector, Value`*)

Parametry

Selektor (VARIANT)-wejście
Selektor elementów danych, które mają być uwzględnione w liczbie.

Wartość MQSEL_ALL_USER_SELECTORS jest wartością domyślną.

Jeśli wartość `Selector` nie jest typu long, zwracana jest wartość MQRC_SELECTOR_TYPE_ERROR.

Wartość (LONG)-dane wyjściowe
Liczba elementów w torbie uwzględnionych przez *`Selector`*.

Wizualne wywołanie języka podstawowego

Aby zwrócić liczbę elementów w torbie:

```
ItemCount = mqbag.Count([Selector])
```

Właściwość opcji

Przeznaczenie

Właściwość Opcje ustawia opcje dla użycia torby. Ta właściwość odpowiada parametrowi **`Options`** wywołania MQAI, "mqCreateBag," w Skorowidz formatów komend programowalnych.

Format

Opcje (*`Options`*)

Parametry

Opcje (LONG)-wejście/wyjście
Opcje torby.

Uwaga: Opcje torby muszą być ustawione jako ***przed*** elementy danych są dodawane do lub ustawiane w worku. Jeśli opcje są zmieniane, gdy worek nie jest pusty, wyniki MQRC_OPTIONS_ERROR są wyświetlane. Ma to zastosowanie nawet wtedy, gdy worek jest następnie wyczyszczony.

Wizualne wywołanie języka podstawowego

Po pytaniu o opcje elementu w obrębie torby:

```
Options = mqbag.Options
```

Aby ustawić opcję elementu w worku:

```
mqbag.Options = Options
```

Metody MQBag

Metody obiektów MQBag są wyjaśnione na następujących stronach.

Dodaj metodę

Przeznaczenie

Metoda Add dodaje element danych do torby. Ta metoda odpowiada wywołaniach MQAI, "mqAddInteger" i "mqAddString" w Skorowidz formatów komend programowalnych.

Format

Dodaj (*Value, Selector*)

Parametry

Value (VARIANT)-wejście
Liczba całkowita lub łańcuchowa elementu danych.

Selektor (VARIANT)-wejście
Selektor identyfikujący element, który ma zostać dodany.

W zależności od typu produktu `Value`wartością domyślną jest MQIA_LIST lub MQCA_LIST. Jeśli parametr **`Selector`** nie jest typu long, wyniki MQRC_SELECTOR_TYPE_ERROR.

Wizualne wywołanie języka podstawowego

Aby dodać element do torby:

```
mqbag.Add(Value,[Selector])
```

Metoda AddInquiry

Przeznaczenie

Metoda AddInquiry dodaje selektor określający atrybut, który ma zostać zwrócony po wysłaniu worka administracyjnego w celu wykonania komendy INQUIRE. Ta metoda odpowiada wywołaniu MQAI, "mqAddInquiry", w Skorowidz formatów komend programowalnych.

Format

AddInquiry *(Inquiry*)

Parametry

Zapytanie (LONG)-dane wejściowe
Selektor atrybutu IBM MQ , który ma zostać zwrócony przez komendę administracyjną INQUIRE.

Wizualne wywołanie języka podstawowego

Aby skorzystać z metody AddInquiry :

```
mqbag.AddInquiry(Inquiry)
```

Wyczyść metodę

Przeznaczenie

Metoda Clear usuwa wszystkie elementy danych z torby. Ta metoda odpowiada wywołaniu MQAI, "mqClearBag,", w Skorowidz formatów komend programowalnych.

Format

Wyczyść

Wizualne wywołanie języka podstawowego

Aby usunąć wszystkie dane itmes z torby:

```
mqbag.Clear
```

Metoda execute

Przeznaczenie

Metoda Execute wysyła komunikat komendy administracyjnej do serwera komend i oczekuje na wszystkie komunikaty odpowiedzi. Ta metoda odpowiada wywołaniu MQAI, "mqExecute", w Skorowidz formatów komend programowalnych.

Format

Wykonaj *(QueueManager, Command, OptionsBag, RequestQ, ReplyQ, ReplyBag*)

Parametry

QueueManager (MQQueueManager)-dane wejściowe
Menedżer kolejek, z którym połączona jest aplikacja.

Komenda (LONG)-dane wejściowe
Komenda do wykonania.

OptionsBag (MQBag)-dane wejściowe
Torba zawierająca opcje, które wpływają na przetwarzanie wywołania.

RequestQ (MQQueue)-dane wejściowe
Kolejka, w której zostanie umieszczony komunikat komendy administracyjnej.

ReplyQ (MQQueue)-dane wejściowe
Kolejka, w której odbierane są wszystkie komunikaty odpowiedzi.

ReplyBag (MQBag)-dane wyjściowe
Odwołanie do torby zawierające dane z komunikatów odpowiedzi.

Wizualne wywołanie języka podstawowego

Aby wysłać komunikat komendy administracyjnej i poczekać na wszystkie komunikaty odpowiedzi:

```
Set ReplyBag = mqbag.Execute(QueueManager, Command,
[OptionsBag],[RequestQ],[ReplyQ])
```

Metoda FromMessage

Przeznaczenie

Metoda FromMessage ładuje dane z wiadomości do torby. Ta metoda odpowiada wywołaniu MQAI, "mqBufferToBag," w Skorowidz formatów komend programowalnych.

Format

FromMessage *(Message, OptionsBag*)

Parametry

Komunikat (MQMessage)-dane wejściowe
Komunikat zawierający dane, które mają zostać przekształcone.

OptionsBag (MQBag)-dane wejściowe
Opcje służące do sterowania przetwarzaniem wywołania.

Wizualne wywołanie języka podstawowego

Aby załadować dane z wiadomości do torby:

```
mqbag.FromMessage(Message,[OptionsBag])
```

Metoda ItemType

Przeznaczenie

Metoda ItemType zwraca typ wartości w określonej pozycji w worku. Ta metoda odpowiada wywołaniu MQAI, "mqInquireItemInfo" w Skorowidz formatów komend programowalnych.

Format

ItemType (*Selector, ItemIndex, ItemType*)

Parametry

Selektor (VARIANT)-wejście
Selektor identyfikujący element, który ma zostać zapytany.

Wartość MQSEL_ANY_USER_SELECTOR jest wartością domyślną. Jeśli parametr **Selector** nie jest typu long, wyniki MQRC_SELECTOR_TYPE_ERROR.

ItemIndex (LONG)-dane wejściowe
Indeks pozycji do zapytania.

Wartość MQIND_NONE jest wartością domyślną.

ItemType (LONG)-dane wyjściowe
Typ danych określonego elementu.

Uwaga: Należy podać parametr **Selector** , parametr **ItemIndex** lub oba te parametry. Jeśli żaden z tych parametrów nie jest obecny, wyniki komendy MQRC_PARAMETER_MISSING.

Wizualne wywołanie języka podstawowego

Aby zwrócić typ wartości:

```
ItemType = mqbag.ItemType([Selector],
[ItemIndex])
```

metoda usuwania

Przeznaczenie

Metoda usuwania usuwa element z torby. Ta metoda odpowiada wywołaniu MQAI, "mqDeleteItem," w Skorowidz formatów komend programowalnych.

Format

Usuń *(Selector, ItemIndex)*

Parametry

Selektor (VARIANT)-wejście
Selektor identyfikujący element, który ma zostać usunięty.

Wartość MQSEL_ANY_USER_SELECTOR jest wartością domyślną. Jeśli parametr **Selector** nie jest typu long, wyniki MQRC_SELECTOR_TYPE_ERROR.

ItemIndex (LONG)-dane wejściowe
Indeks elementu, który ma zostać usunięty.

Wartość MQIND_NONE jest wartością domyślną.

Uwaga: Należy podać parametr **Selector** , parametr **ItemIndex** lub oba te parametry. Jeśli żaden z tych parametrów nie jest obecny, wyniki komendy MQRC_PARAMETER_MISSING.

Wizualne wywołanie języka podstawowego

Aby usunąć pozycję z torby:

```
mqbag.Remove([Selector],[ItemIndex])
```

Metoda selektora

Przeznaczenie

Metoda Selector zwraca selektor określonego elementu w obrębie torby. Ta metoda odpowiada wywołaniu MQAI, "mqInquireItemInfo" w Skorowidz formatów komend programowalnych.

Format

Selektor *(Selector, ItemIndex, OutSelector)*

Parametry

Selektor (VARIANT)-wejście
Selektor identyfikujący element, który ma zostać zapytany.

Wartość MQSEL_ANY_USER_SELECTOR jest wartością domyślną. Jeśli parametr **Selector** nie jest typu long, wyniki MQRC_SELECTOR_TYPE_ERROR.

ItemIndex (LONG)-dane wejściowe
Indeks elementu, który ma zostać sprawdzony.

Wartość MQIND_NONE jest wartością domyślną.

OutSelector (VARIANT)-dane wyjściowe
Selektor określonego elementu.

Uwaga: Należy podać parametr **Selector** , parametr **ItemIndex** lub oba te parametry. Jeśli żaden z tych parametrów nie jest obecny, wyniki komendy MQRC_PARAMETER_MISSING.

Wizualne wywołanie języka podstawowego

Aby zwrócić selektor elementu:

```
OutSelector = mqbag.Selector([Selector],
[ItemIndex])
```

Metoda ToMessage

Przeznaczenie

Metoda ToMessage zwraca odwołanie do obiektu MQMessage. Odwołanie zawiera dane z torby. Ta metoda odpowiada wywołaniu MQAI, "mqBagToBuffer," w Skorowidz formatów komend programowalnych.

Format

ToMessage *(OptionsBag, Message)*

Parametry

OptionsBag (MQBag)-dane wejściowe
Torba zawierająca opcje, które sterują przetwarzaniem metody.

Komunikat (MQMessage)-dane wyjściowe
Odwołanie do obiektu MQMessage zawierające dane z worka.

Wizualne wywołanie języka podstawowego

Aby użyć metody ToMessage , wykonaj następujące czynności:

```
Set Message = mqbag.ToMessage([OptionsBag])
```

Metoda obcinania

Przeznaczenie

Metoda obcinania zmniejsza liczbę elementów użytkownika w torbie. Ta metoda odpowiada wywołaniu MQAI, "mqTruncateBag," w Skorowidz formatów komend programowalnych.

Format

Obetnij *(ItemCount)*

Parametry

ItemCount (LONG)-dane wejściowe
Liczba elementów użytkownika, które mają pozostać w torbie po obcięciu.

Wizualne wywołanie języka podstawowego

Aby zmniejszyć liczbę elementów użytkownika w torbie:

```
mqbag.Truncate(ItemCount)
```

Informacje o klasach automatyzacji produktu IBM MQ dla przykładów startowych ActiveX

W niniejszym dodatku opisano klasy automatyzacji produktu IBM MQ dla przykładów startowych ActiveX oraz objaśniono, w jaki sposób można ich używać.

Produkt IBM MQ for Windows udostępnia następujące przykładowe programy Visual Basic:

- MQAXTRIV.VBP

- MQAXBSRV.VBP
- MQAXDLST.VBP
- MQAXCLSS.VBP

Te przykłady są uruchamiane w Visual Basic 4 lub Visual Basic 5. Znajdą je Państwo w katalogu ... \tools\mqax\samples\vb.

W tym samym katalogu znajdują się również przykłady dla programu Microsoft Excel i html. Są to:

- MQAX.XLS
- MQAXTRIV.XLS
- MQAXTRIV.HTM

Uwaga: Jeśli używany jest produkt Visual Basic 5, **należy** wybrać i zainstalować komponent Visual Basic grid32.ocx.

Co wykazano w próbkach

W przykładach przedstawiono sposób użycia klas automatyzacji produktu IBM MQ dla elementu ActiveX do:

- Połącz z menedżerem kolejek
- Dostęp do kolejki
- Umieszczanie komunikatu w kolejce
- Pobieranie komunikatu z kolejki

Centralna część przykładu Visual Basic jest pokazana na kolejnych stronach.

"Przygotowanie do uruchomienia przykładów" na stronie 777 i

"Obsługa błędów w próbkach" na stronie 777

Uruchamianie przykładów ActiveX Starter

Przed uruchomieniem klas automatyzacji produktu IBM MQ dla ActiveX Starter należy sprawdzić, czy domyślny menedżer kolejek jest uruchomiony i czy zostały utworzone wymagane definicje kolejek. Szczegółowe informacje na temat tworzenia i uruchamiania menedżera kolejek oraz tworzenia kolejki można znaleźć w sekcji Administrowanie. W tym przykładzie używana jest kolejka SYSTEM.DEFAULT.LOCAL.QUEUE , która powinna być zdefiniowana na dowolnym normalnie ustawionym serwerze IBM MQ .

Różne sposoby korzystania z toreb danych są przedstawione na poniższej liście:

- Połącz z menedżerem kolejek
- Dostęp do kolejki
- Umieszczanie komunikatu w kolejce
- Pobieranie komunikatu z kolejki

Informacje na temat przykładów startowych MQAX dla produktu Microsoft w wersji Basic 4 lub nowszej zawiera sekcja "Uruchamianie przykładu MQAXTRIV" na stronie 777

Informacje na temat przykładu, które umożliwiają przeglądanie właściwości i metod menedżerów kolejek i obiektów kolejki, zawiera sekcja "Uruchamianie przykładu MQAXCLSS" na stronie 779 .

For information on the MQAXDLST sample, "Przykład MQAXDLST" na stronie 779

Informacje na temat uruchamiania przykładu startowego MQAX dla programu Microsoft Excel 95 lub nowszego, MQAXTRIV.XLS, patrz "Uruchamianie komendy MQAXTRIV.XLS , przykład" na stronie 779.

Informacje na temat uruchamiania demonstracji Banku za pomocą programu MQAX.XLS, patrz "Uruchamianie demonstracji w Banku za pomocą programu MQAX.XLS" na stronie 779 .

Informacje na temat przykładowego programu startowego za pomocą przeglądarki WWW kompatybilnej z ActiveX można znaleźć w sekcji “Przykład startowy za pomocą przeglądarki WWW kompatybilnej z ActiveX” na stronie 779 .

Przygotowanie do uruchomienia przykładów

Aby uruchomić dowolną z przykładów, należy wykonać jedną z poniższych czynności w zależności od tego, które z przykładów mają być uruchamiane.

- Microsoft Visual Basic 4 (lub nowszy)
- Microsoft Excel 95 (lub nowszy)
- Przeglądarka WWW

Potrzebne są również:

- Menedżer kolejek produktu IBM MQ jest uruchomiony.
- Kolejka IBM MQ jest już zdefiniowana.

Obsługa błędów w próbkach

Większość przykładów dostępnych w pakiecie IBM MQ Automation Classes for ActiveX nie ma możliwości obsługi błędów lub nie jest w tym przypadku. Więcej informacji na temat obsługi błędów zawiera sekcja “Obsługa błędów” na stronie 691.

Uruchamianie przykładu MQAXTRIV

1. Uruchom menedżer kolejek.
2. W Eksploratorze Windows lub File Managerwybierz ikonę dla przykładu MQAXTRIV.VBP (plik projektu Visual Basic) i otwórz plik.

   Program Visual Basic uruchamia i otwiera plik MQAXTRIV.VBP.
3. W Visual Basic naciśnij klawisz funkcyjny 5 (F5), aby uruchomić próbkę.
4. Kliknij dowolne miejsce w formularzu okna **MQAX trivial tester**(tester trójfiolek MQAX).

Jeśli wszystko działa poprawnie, tło okna powinno zmienić się na zielony. Jeśli wystąpił problem z konfiguracją, tło okna powinno zostać zmienione na czerwony, a informacje o błędach zostaną wyświetlone.

Na poniższym rysunku przedstawiono centralną część przykładu Visual Basic.

```
Option Explicit

Private Sub Form_Click()

'*************************************************************************
'* This simple example illustrates how to put and get an IBM MQ message to
'* and from an IBM MQ message queue. The data from the message returned by the
'* get is read and compared with that from the original message.
'*************************************************************************
Dim MQSess As MQSession                 '* session object
Dim QMgr As MQQueueManager              '* queue manager object
Dim Queue As MQQueue                    '* queue object
Dim PutMsg As MQMessage                 '* message object for put
Dim GetMsg As MQMessage                 '* message object for get
Dim PutOptions As MQPutMessageOptions   '* put message options
Dim GetOptions As MQGetMessageOptions   '* get message options
Dim PutMsgStr As String                 '* put message data string
Dim GetMsgStr As String                 '* get message data string
'*************************************************************************
'* Handle errors
'*************************************************************************
On Error GoTo HandleError

'*************************************************************************
```

```
'* Initialize the current position for the form
'*******************************************************************************
CurrentX = 0
CurrentY = 0

'*******************************************************************************
'* Create the MQSession object and access the MQQueueManager and (local) MQQueue
'*******************************************************************************
Set MQSess = New MQSession
Set QMgr = MQSess.AccessQueueManager("")
Set Queue = QMgr.AccessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE", _
                      MQOO_OUTPUT Or MQOO_INPUT_AS_Q_DEF)

'*******************************************************************************
'* Create a new MQMessage object for use with put, add some data then create an
'* MQPutMessageOptions object and put the message
'*******************************************************************************
Set PutMsg = MQSess.AccessMessage()
PutMsgStr = "12345678 " & Time
PutMsg.MessageData = PutMsgStr
Set PutOptions = MQSess.AccessPutMessageOptions()
Queue.Put PutMsg, PutOptions

'*******************************************************************************
'* Create a new MQMessage object for use with get, set the MessageId (to that of
'* the message that was put), create an MQGetMessageOptions object and get the
'* message.
'*
'* Note: Setting the MessageId ensures that the get returns the MQMessage
'* that was put earlier.
'*******************************************************************************

Set GetMsg = MQSess.AccessMessage()
GetMsg.MessageId = PutMsg.MessageId
Set GetOptions = MQSess.AccessGetMessageOptions()
Queue.Get GetMsg, GetOptions
'*******************************************************************************
'* Read the data from the message returned by the get, compare it with
'* that from the original message and output a suitable message.
'*******************************************************************************
GetMsgStr = GetMsg.MessageData
Cls
If GetMsgStr = PutMsgStr Then
    BackColor = RGB(127, 255, 127) '* set to green for ok
    Print
    Print "Message data comparison was successful."
    Print "Message data: """ & GetMsgStr & """"

Else
    BackColor = RGB(255, 255, 127)  '* set to amber for compare error
    Print "Compare error: "
    Print "The message data returned by the get did not match the " &_
    "input data from the original message that was put."
    Print
    Print "Input message data:        """ & PutMsgStr & """"
    Print "Returned message data: """ & GetMsgStr & """"
End If

Exit Sub
'*******************************************************************************
'* Handle errors
'*******************************************************************************
HandleError:
Dim ErrMsg As String
Dim StrPos As Integer

Cls
BackColor = RGB(255, 0, 0)  '* set to red for error
Print "An error occurred as follows:"
Print ""
If MQSess.CompletionCode <> MQCC_OK Then
    ErrMsg = Err.Description
    StrPos = InStr(ErrMsg, " ")            '* search for first blank
    If StrPos > 0 Then
        Print Left(ErrMsg, StrPos)         '* print offending MQAX object name
    Else
        Print Error(Err)                   '* print complete error object
    End If
    Print ""
    Print "IBM MQ Completion Code = " & MQSess.CompletionCode
    Print "IBM MQ Reason Code = " & MQSess.ReasonCode
    Print "(" & MQSess.ReasonName & ")"
```

```
Else
    Print "Visual Basic error: " & Err
    Print Error(Err)
End If

Exit Sub

End Sub
```

Uruchamianie przykładu MQAXCLSS

Ten przykład umożliwia przeglądanie właściwości i metod menedżerów kolejek i obiektów kolejki.

1. Uruchom menedżer kolejek.
2. Otwórz plik MQAXCLSS.VBP, klikając dwukrotnie ikonę dokumentu w Eksploratorze Windows lub klikając opcję Plik-Otwórz z menu Plik w języku Visual Basic.
3. Uruchom przykład.
4. Wprowadź odpowiednie nazwy menedżerów kolejek i kolejek, a następnie kliknij odpowiednie przyciski.

Przykład MQAXDLST

Przykład Visual Basic MQAXDLST demonstruje użycie listy dystrybucyjnej w celu wysłania tego samego komunikatu do dwóch kolejek przy użyciu jednej operacji put. Aby uruchomić przykład, należy wykonać to samo, co w przypadku przykładu MQAXCLSS.

Przykład MQAX Starter dla programu Microsoft Excel 95 lub nowszego

W tej sekcji opisano sposób uruchamiania przykładu startowego MQAX dla programu Microsoft Excel 95 lub nowszego, MQAXTRIV.XLS.

Uruchamianie komendy MQAXTRIV.XLS , przykład

1. Uruchom menedżer kolejek.
2. W Eksploratorze lub File Managerwybierz ikonę dla przykładowego programu MQAX MQAXTRIV.XLS.
3. Kliknij przycisk w arkuszu kalkulacyjnym.
4. Ekran zostanie zaktualizowany za pomocą komunikatu o powodzeniu (lub niepowodzeniu).

Uruchamianie demonstracji w Banku za pomocą programu MQAX.XLS

Wykonaj poniższe kroki, aby uruchomić demonstracę Banku.

1. Uruchom menedżer kolejek.
2. Uruchom plik komend MQSC IBM MQ , BANK.TST. Spowoduje to ustawienie niezbędnych definicji kolejek produktu IBM MQ .

   Aby dowiedzieć się, jak używać pliku komend MQSC, należy zapoznać się z administracją za pomocą komend MQSC.
3. Uruchom komendę MQAXBSRV.VBP. Ten przykładowy program to serwer symulujący aplikację zaplecza, który musi być uruchomiony z programem Microsoft Excel.
4. Uruchom program MQAX.XLS. Ten przykład jest demonstracją klienta IBM MQ .
5. Wybierz klienta z listy.
6. Kliknij przycisk **Submit** (Wyślij).

Po krótkiej przerwie (około 3 sekundy) wyświetlane są pola zapełniania wartościami i wykresem słupkowym.

Przykład startowy za pomocą przeglądarki WWW kompatybilnej z ActiveX

Uwaga: Aby uruchomić ten przykład, należy uruchomić kompatybilną przeglądarkę WWW ActiveX . Microsoft Edge (ale nie Netscape Navigator) jest kompatybilną przeglądarką WWW.

Uruchamianie przykładu HTML

W tym przykładzie pokazano, w jaki sposób można wywołać program MQAX zarówno z języka VBScript, jak i z produktu JavaScript.

1. Uruchom menedżer kolejek.
2. Otwórz plik "MQAXTRIV.HTM", w przeglądarce WWW zgodnej z ActiveX .

   Można to zrobić, klikając dwukrotnie ikonę pliku w programie Windows Explorer lub wybierając opcję Plik-Otwórz z menu Plik przeglądarki WWW kompatybilnej z ActiveX .
3. Postępuj zgodnie z instrukcjami wyświetlanym na ekranie.

ULW Tworzenie aplikacji klienckich AMQP

Obsługa produktu IBM MQ dla interfejsów API AMQP, w tym interfejsu API produktu MQ Light , 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 systemie UNIX, Linuxlub Windows. Nie jest on dostępny w systemach IBM i i z/OS.

Interfejs API MQ Light jest oparty na protokole Oasis AMQP 1.0 . Dostępne są interfejsy API przesyłania komunikatów dla Node.js, Java, Ruby i Python.

Aplikacja, która została zaprojektowana z myślą o korzystaniu z funkcji API języka MQ Light , może być połączona ze środowiskiem wykonawczym produktu MQ Light , menedżerem kolejek produktu IBM MQ z kanałem AMQP lub instancją usługi produktu MQ Light w produkcie IBM Cloud (formerly Bluemix).

Tworzenie klientów AMQP

Interfejs API MQ Light ma na celu ułatwienie tworzenia prototypów i szybkiego tworzenia aplikacji biznesowych. Dostępne są interfejsy API MQ Light dla Node.js, Java, Ruby i Python, dostępne pod adresem https://github.com/mqlight.

Pobieranie przykładowych klientów AMQP

IBM MQ nie jest dostarczane z klientami MQ Light , ale można pobrać i zainstalować następujące klienty MQ Light :

Node.js
Zainstaluj interfejs API MQ Light Node.js w katalogu roboczym, używając npm: `npm install mqlight@1.0`

Java
Pobierz pakiet mqlight-distribution dla wymaganej wersji z serwisu Maven Central i rozpakuj jego zawartość. Dostępne wersje pakietów dystrybucyjnych mqlight można znaleźć w serwisie Maven Central.

Ruby
Zainstaluj interfejs MQ Light Ruby API w katalogu roboczym, używając gem: `gem install mqlight --pre`

Python
Zainstaluj interfejs API języka MQ Light Python w katalogu roboczym, używając programu pip: `pip install mqlight --pre`

Wszystkie pliki do pobrania klienta MQ Light zawierają kilka przykładów, które przedstawiają różne funkcje przesyłania komunikatów:

- Wyślij przykład
- Odbierz przykład
- Przykład treningu interfejsu użytkownika

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.

Zabezpieczanie klientów AMQP

Informacje na temat zabezpieczania aplikacji MQ Light zawiera sekcja Zabezpieczanie klientów AMQP.

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. Aplikacje MQ Light można wdrożyć w menedżerze kolejek systemu IBM MQ . Patrz “Wdrażanie aplikacji MQ Light w środowisku IBM MQ w siedzibie” na stronie 799 .

Po wdrożeniu klienta AMQP można wymieniać komunikaty z aplikacjami IBM MQ . Jeśli na przykład klient MQ Light Node.js jest używany do wysyłania komunikatu łańcuchowego JavaScript , aplikacja IBM MQ odbiera komunikat 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 aplikacją MQ Light przy użyciu dowolnej z następujących metod:

- Używanie klienta IBM MQ Light dla Node.js i Java.
- Korzystanie z klienta IBM MQ Light dla języków Ruby i Python.
- Użycie innego klienta AMQP 1.0 . Na przykład Apache Qpid Proton.

Zadania pokrewne

Tworzenie i używanie kanałów AMQP

Zabezpieczanie klientów AMQP

ULW MQ Light i AMQP (Advanced Message Queuing Protocol)

Interfejs API IBM MQ Light jest oparty na protokole OASIS Standard AMQP 1.0 . Program AMQP określa sposób wysyłania komunikatów między nadawcami i odbiorcami. Aplikacja działa jako nadawca, gdy aplikacja wysyła komunikat do brokera komunikatów, takiego jak IBM MQ. Produkt IBM MQ działa jako nadawca, gdy wysyła komunikat do aplikacji AMQP.

Niektóre z korzyści płynnych z AMQP są następujące:

- Otwarty protokół standaryzowany
- Kompatybilność z innymi klientami Open Source AMQP 1.0
- Dostępnych jest wiele implementacji klienta Open Source.

Mimo że 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 oraz OASIS Standard AMQP 1.0 PDF.

Interfejs API przesyłania komunikatów produktu MQ Light jest oparty na produkcie AMQP 1.0. Interfejs API udostępnia większość możliwości przesyłania komunikatów, które są wymagane przez większość przepływów przesyłania komunikatów w trybie publikowania/subskrypcji i w trybie punkt z punktem.

Interfejs API produktu MQ Light zawiera następujące funkcje przesyłania komunikatów:

- Dostarczanie komunikatów w większości po raz
- Dostarczanie komunikatów przynajmniej raz
- Adresowanie miejsca docelowego łańcucha tematu
- Trwałość komunikatu i miejsca przeznaczenia
- Współużytkowane miejsca docelowe w celu umożliwienia wielu subskrybentom współużytkowania obciążenia
- Przejęcie klienta w celu łatwego rozwiązania zawieszonych klientów
- Konfigurowalny odczyt z wyprzedzeniem komunikatów
- Konfigurowalne potwierdzenie komunikatów

Kompletną dokumentację interfejsu API produktu MQ Light można znaleźć w następujących serwisach WWW:

- Dokumentację funkcji API Node.js można znaleźć w sekcji https://www.npmjs.org/package/mqlight .
- Dokumentację interfejsu API języka Ruby można znaleźć w sekcji https://www.rubydoc.info/github/mqlight/ruby-mqlight
- Dokumentację funkcji API języka Python można znaleźć w sekcji https://python-mqlight.readthedocs.org
- Dokumentację interfejsu API produktu Java można znaleźć w sekcji https://mqlight.github.io/java-mqlight

Zadania pokrewne

Tworzenie kanałów AMQP i korzystanie z nich

Zabezpieczanie klientów AMQP

ULW Obsługa AMQP 1.0

Kanały AMQP zapewniają poziom obsługi dla aplikacji zgodnych ze standardem AMQP 1.0-compliant .

Kanały AMQP obsługują podzbiór protokołu AMQP 1.0 . Do kanału AMQP produktu IBM MQ można połączyć klienty MQ Light lub inne kompatybilne klienty AMQP 1.0 . Aby korzystać ze wszystkich funkcji przesyłania komunikatów obsługiwanych przez kanały AMQP, należy poprawnie ustawić wartość niektórych pól AMQP 1.0 .

Te informacje zawierają informacje na temat sposobu formatowania pól AMQP i zawiera listę składników specyfikacji AMQP 1.0 , które nie są obsługiwane przez kanały AMQP.

Następujące funkcje specyfikacji AMQP 1.0 nie są obsługiwane lub są ograniczone w ich użyciu:

Nazwy odsyłaczy

Kanały AMQP oczekują, że nazwa łącza AMQP będzie zgodna z jednym z trzech formatów:

- Prosty temat (w celu 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 szczegółowy temat (w przypadku subskrybowania)

  - Szczegółowy łańcuch tematu opisujący subskrypcję prywatną w formularzu: "`private:`*`topic string`*" (na przykład: "`private:/sports/football`"). Zachowanie jest takie samo jak zwykły łańcuch tematu. Deklaracja `private` różnicuje subskrypcję specyficzną dla konkretnego klienta AMQP z subskrypcji współużytkowanej między klientami.
- Współużytkowany temat szczegółowy (w przypadku subskrybowania)
  - Szczegółowy łańcuch tematu opisujący subskrypcję współużytkowaną w formularzu: "`share:`*`share name`*`:`*`topic string`*" (na przykład: "`share:bbc:/sports/football`").

Więcej informacji na temat sposobu odwzorowywania komunikatów AMQP na i z komunikatów produktu IBM MQ zawiera sekcja Odwzorowywanie pól AMQP na pola produktu IBM MQ (komunikaty przychodzące).

Maksymalna długość łańcuchów tematów, nazw zasobów współużytkowanych i identyfikatorów klientów

Łańcuch tematu, nazwa zasobu współużytkowanego i identyfikator klienta muszą być zawarte w ciągu 10237 bajtów. Ponadto maksymalna długość identyfikatora klienta wynosi 256 znaków.

Te maksymalne długości oznaczają, że użytkownik może mieć jedną z następujących wartości:

- 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 performatywnego typu AMQP Open będzie zawierał unikalny identyfikator klienta MQ Light . Maksymalna długość identyfikatora klienta MQ Light to 256 znaków, a identyfikator może zawierać znaki alfanumeryczne, znak procentu (%), ukośnik (/), kropkę (.) i znak podkreślenia (_).

Sesje

Kanały AMQP obsługują tylko pojedynczą 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 dołączania AMQP, która próbuje skoordynować nową transakcję lub ramkę przesyłania AMQP, która próbuje zadeklarować nową transakcję, została odrzucona z komunikatem o błędzie.

Stan dostawy

Kanały AMQP obsługują tylko stan dostawy dla ramek dyspozycji zaakceptowanych.

Zadania pokrewne

Tworzenie kanałów AMQP i korzystanie z nich

Zabezpieczanie klientów AMQP

ULW Odwzorowywanie pól komunikatów AMQP i IBM MQ

Komunikaty AMQP składają się z nagłówka, adnotacji dostarczania, adnotacji komunikatów, właściwości, właściwości aplikacji, treści i stopki.

Komunikaty AMQP składają się z następujących części:

Nagłówek

Nagłówek opcjonalny 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, wiadomość nie została przejęta przez żaden inny odsyłacz.
- **count-count** -liczba poprzednich, niepomyślnych prób dostarczenia.

Dostarczanie-adnotacje
Opcjonalne. Określa niestandardowe atrybuty nagłówka komunikatu dla różnych odbiorców. Adnotacje dostarczania przekazują informacje z węzła wysyłającego do węzła odbierającego.

Komunikaty-adnotacje
Opcjonalne. Określa niestandardowe atrybuty nagłówka komunikatu dla różnych odbiorców. Sekcja adnotacje komunikatu jest używana dla właściwości komunikatu, które mają być skierowane do infrastruktury i powinny być propagowane w każdym kroku dostarczania.

Właściwości
Opcjonalne. Ta część jest odpowiednikiem deskryptora komunikatu produktu MQ . Zawiera następujące stałe pola:

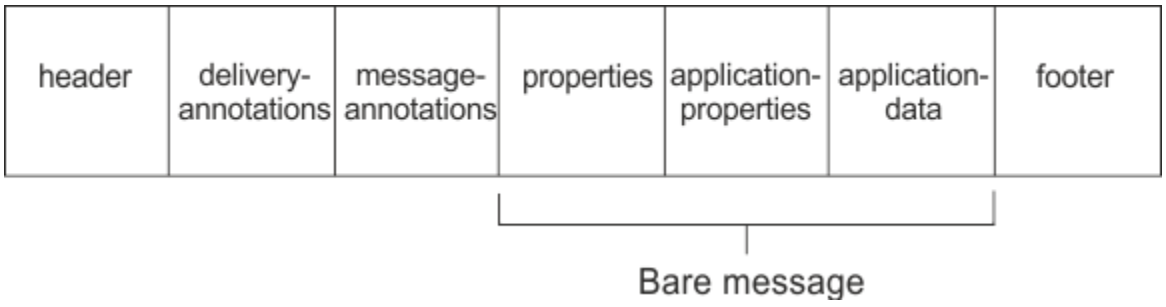
- **message-id** -identyfikator komunikatu aplikacji
- **user-id** -identyfikator użytkownika tworzący
- **to** -adres węzła, dla którego komunikat jest przeznaczony
- **subject** (temat)-temat komunikatu
- **reply-to** -węzeł, do którego wysyłane są odpowiedzi.
- **correlation-id** -identyfikator korelacji aplikacji
- **content-type** -typ treści MIME
- **content-encoding** -typ treści MIME. Używany jako modyfikator do typu treści.
- **absolute-upłynął-czas** -czas, w którym ten komunikat został uznany za utracony.
- **creation-time** -czas utworzenia tego komunikatu.
- **group-id** -grupa, do której należy ten komunikat
- **group-sequence** -numer kolejny tego komunikatu w obrębie grupy.
- **reply-to-group-id** -grupa, do której należy komunikat odpowiedzi

Aplikacje-właściwości
Odpowiednik właściwości komunikatu produktu MQ .

Treść
Odpowiednik ładunku użytkownika MQ .

Stopka
Opcjonalne. Stopka jest używana w celu uzyskania szczegółowych informacji o komunikacie lub dostarczeniu, które mogą być obliczane lub wartościowane tylko po skonstruowaniu lub obejrzaniu całego gołego komunikatu (na przykład hashów komunikatu, HMACs, sygnatur i szczegółów szyfrowania).

Format komunikatu AMQP jest ilustrowany na poniższym rysunku:



Właściwości, właściwości application-properties i application-data są nazywane "samym komunikatem". Jest to komunikat wysyłany przez nadawcę i jest on niezmienny. Odbiornik widzi cały komunikat, w tym nagłówek, stopkę, adnotacje dostarczania i adnotacje komunikatów.

Pełny opis formatu komunikatu AMQP 1.0 można znaleźć w normie OASIS na stronie https://docs.oasis-open.org/amqp/core/v1.0/amqp-core-complete-v1.0.pdf.

Zadania pokrewne

Tworzenie kanałów AMQP i korzystanie z nich

Zabezpieczanie klientów AMQP

ULW Odwzorowywanie pól produktu IBM MQ na pola AMQP (komunikaty wychodzące)

Gdy zostanie opublikowany komunikat IBM MQ , a program IBM MQ wysyła go do konsumenta AMQP, propaguje niektóre atrybuty komunikatu produktu IBM MQ do równoważnych atrybutów komunikatu AMQP.

header (nagłówek)

Nagłówek jest uwzględniany tylko wtedy, gdy jedno z pięciu pól w nagłówku zawiera wartość inną niż domyślna. W nagłówku uwzględniane są tylko pola z wartością inną niż domyślna. Pięć pól nagłówka pochodzi początkowo z równoważnej właściwości `mq_amqp.Hdr` , jeśli jest ona ustawiona, a następnie modyfikowana, jak pokazano w poniższej tabeli:

Tabela 105. Odwzorowania pól nagłówka

Pole	Wartość domyślna	Wartość
Trwałe	Fałsz	Wartość true, jeśli parametr `MQMD.Persistence` ma wartość `MQPER_PERSISTENT`, w przeciwnym razie wartość false.
priorytet	4	Od `mq_amqp.Hdr.Pri`, jeśli jest ustawiony, lub w inny sposób z `MQMD.Priority`, jeśli jest ustawiony. Jeśli żaden z nich nie jest ustawiony, ustaw wartość 4.
ttl	nie dotyczy	`MQMD.Expiry` (w milisekundach). Jeśli wartością parametru `MQMD.Expiry` jest `MQEI_UNLIMITED`, należy ustawić wartość maksymalną dla pola ttl AMQP.
pierwszy nabywca	Fałsz	W przypadku ustawienia `mq_amqp.Hdr.Fac`(jeśli jest ustawiona) lub false (fałsz).
liczba pobrań	0	Od `mq_amqp.Hdr.Dct`, jeśli jest ustawiony, lub 0 w przeciwnym razie.

adnotacja dostarczania

Ustaw zgodnie z tym, co jest konieczne przez kanał AMQP.

adnotacja komunikatu

Nie uwzględniono.

właściwości

Jeśli te właściwości zostaną ustawione, **właściwości** nie zostaną zmodyfikowane z równoważnych właściwości produktu `mq_amqp.Prp` . Jeśli komunikat nie był pierwotnie komunikatem AMQP (czyli PutAppl, typ nie jest typu MQAT_AMQP), wówczas zostanie wygenerowana sekcja właściwości zgodnie z opisem w poniższej tabeli:

Tabela 106. Odwzorowania pól właściwości

Nazwa	**Wartość**
id-komunikatu	`MQMD.MsgId` jest ustawiony jako binarny.
id-użytkownika	Formularz UTF-8 w `MQMD.UserIdentifier` jest ustawiony jako binarny w kolejności bajtów w sieci.
do	Kolejka, z której otrzymano komunikat, lub, w przypadku publikacji, łańcuch tematu.
obiekt	Nie ustawiono.
odpowiedź-na	Wartość `MQMD.ReplyToQ` , jeśli nie jest pusta, w przeciwnym razie nie jest ustawiona.
Identyfikator korelacji	Wartość `MQMD.CorrelId` jest ustawiana jako binarna, jeśli nie jest pusta, w przeciwnym razie nie jest ustawiona.
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 datownika.
id grupy	Nie ustawiono.
grupa-sekwencja	Nie ustawiono.
id-grupy-odpowiedzi	Nie ustawiono.

właściwości aplikacji

Wszystkie właściwości produktu IBM MQ w grupie "usr" są dodawane jako **application-properties**(właściwości aplikacji).

treść

Kanał AMQP wykonuje proces get z konwersję, aby przekształcić ładunek IBM MQ w format UTF-8.

Jeśli ładunek IBM MQ nie zawiera komunikatu AMQP, ładunek IBM MQ jest ustawiany w treści jako pojedyncza sekcja danych łańcucha dla formatu MQFMT_STRING (udostępniona konwersja do UTF-8 powiodła się) lub jako pojedyncza binarna sekcja danych w przeciwnym razie.

Jeśli dołączany jest komunikat w formacie AMQP, to jest on ustawiany jako treść. Wszystkie nagłówki produktu IBM MQ (bez właściwości komunikatów, które są zwracane w uchwycie komunikatu) poprzedzające komunikat AMQP, są poprzedzane jako wartość binarna, jeśli treść jest sekwencją AMQP. W przeciwnym razie nagłówki IBM MQ zostaną usunięte.

stopka

Nie dołączono stopki.

Zadania pokrewne

Tworzenie kanałów AMQP i korzystanie z nich

Zabezpieczanie klientów AMQP

Odsyłacze pokrewne

MQMD-deskryptor komunikatu

ULW Odwzorowywanie pól AMQP na pola IBM MQ (komunikaty przychodzące)

Gdy kanał AMQP odbiera komunikat i umieszcza go w produkcie IBM MQ, propaguje niektóre atrybuty komunikatu AMQP do równoważnych atrybutów komunikatu produktu IBM MQ .

Podczas odwzorowywania przychodzącego komunikatu AMQP mają zastosowanie następujące ograniczenia:

- Jeśli pole `message-id` lub `correlation-id` w części właściwości jest uuid lub ulong, to komunikat zostaje odrzucony.
- Dowolny `message-annotations` powoduje odrzucenie komunikatu.
- Sekcje `delivery-annotations` i `footer` są dozwolone, ale nie są propagowane do komunikatu produktu IBM MQ .

W poniższych podsekcjach przedstawiono wyrażenie IBM MQ komunikatu AMQP.

deskryptor komunikatu

Tabela 107. Deskryptor komunikatu dla komunikatu AMQP

Pole	**Wartość**
StrucId	MQMD_STRUC_ID
Wersja	MQMD_VERSION_1
Raport	MQRO_NONE
MsgType	MQMT_DATAGRAM
Utrata ważności	Wartość pobierana z pola `ttl` w nagłówku komunikatu AMQP
Opinie	MQFB_NONE
Kodowanie	MQENC_NORMAL
CodedCharSetId	1208 (UTF-8)
Format	Patrz ładunek
Priorytet	Wartość pobierana z pola `priority` w nagłówku komunikatu AMQP. Jeśli jest ustawiona, ograniczona do wartości maksymalnej 9. Jeśli nie jest ustawiony, przyjmuje wartość domyślną 4.
Trwałość	Jeśli wartość pola `durable` w nagłówku komunikatu AMQP jest ustawiona na wartość true, ustaw wartość MQPER_PERSISTENT. W przeciwnym razie ustaw wartość MQPER_NOT_PERSISTENT.
MagId	Menedżer kolejek przydziela unikalny 24-bajtowy identyfikator MsgId.
Correlld	Wartość pobierana z pola `correlation-id` we właściwościach AMQP, jeśli jest ustawiona. Ustawiana jest wartość binarna 24-bajtowa. W przeciwnym razie należy ustawić wartość MQCI_NONE/.
BackoutCount	0
ReplyToQ	""

Tabela 107. Deskryptor komunikatu dla komunikatu AMQP (kontynuacja)	
Pole	**Wartość**
ReplyToQMgr	""
UserIdentifier	Ustaw identyfikator uwierzytelnionego użytkownika, który jest połączony z kanałem AMQP.
AccountingToken	MQACT_NONE
Dane_tożsamości_aplikacji	Łańcuch szesnastkowy. Ustaw na ostatnie 8 bajtów identyfikatora połączenia MQ kanału AMQP.
Typ_aplikacji_wstawiającej	MQAT_AMQP
Nazwa_aplikacji_wstawiającej	
PutDate	Wartość pobierana z pola `creation-time` we właściwościach AMQP, jeśli jest ustawiona. W przeciwnym razie należy ustawić bieżącą datę.
PutTime	Wartość pobierana z pola `creation-time` we właściwościach AMQP, jeśli jest ustawiona. W przeciwnym razie ustaw wartość bieżącą.
Dane_pochodzenia_aplikacji	""

Właściwości komunikatu

Istnieją dwa powody ustawiania właściwości komunikatu:

- Aby umożliwić przepływ części komunikatu AMQP przez menedżer kolejek bez wpływu na ładunek komunikatu.
- Aby zezwolić na wybór `application-properties`.

W poniższej tabeli przedstawiono właściwości, które są ustawiane na podstawie komunikatu AMQP:

Tabela 108. Właściwości komunikatu AMQP			
Nazwa właściwości	**Nazwa MQRFH2**	**Typ**	**Opis**
Program AMQPListener	mq_amqp.Lis	MQTYPE_STRING	Łańcuch identyfikujący kanał AMQP. Jest on używany do generowania komunikatu, aby zainteresowane strony mogły określić, która wersja jest umieszczana w komunikacie (na przykład zespół serwisowy podczas diagnozowania problemów). Wartość nie jest sprawdzana przez menedżera kolejek i nie może być udokumentowana zewnętrznie.
Wersja programu AMQPVersion	mq_amqp.Ver	MQTYPE_STRING	Wersja komunikatu AMQP. Jeśli nie jest obecny, przyjmowany jest wartość "1.0". Poprawność wartości nie jest sprawdzana przez menedżer kolejek.

Tabela 108. Właściwości komunikatu AMQP (kontynuacja)

Nazwa właściwości	Nazwa MQRFH2	Typ	Opis
Klient AMQPClient	mq_amqp.Cli	MQTYPE_STRING	Łańcuch identyfikujący interfejs API. Jest on używany do wysyłania komunikatu AMQP do kanału, tak aby zainteresowane strony mogły określić, która wersja jest umieszczana w komunikacie (na przykład zespół serwisowy podczas diagnozowania problemów). Wartość nie jest sprawdzana przez menedżer kolejek i nie może być udokumentowana zewnętrznie.
{PDurable	mq_amqp.Hdr.Dur	MQTYPE_BOOLEAN	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.
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 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	Wartość pola `message-id` we właściwościach AMQP, jeśli jest ustawiona jako łańcuch.
		MQTYPE_BYTE_STRING	Wartość pola `message-id` we właściwościach AMQP, jeśli jest ustawiona jako łańcuch bajtów.
AMQPUserId	mq_amqp.Prp.Uid	MQTYPE_BYTE_STRING	Wartość pola `user-id` we właściwościach AMQP, jeśli jest ustawiona.
AMQPTo	mq_amqp.Prp.To	MQTYPE_STRING	Wartość pola `to` we właściwościach AMQP, jeśli jest ustawiona.
Obiekt AMQPSubject	mq_amqp.Prp.Sub	MQTYPE_STRING	Wartość pola `subject` we właściwościach AMQP, jeśli jest ustawiona.
AMQPReplyTo	mq_amqp.Prp.Rto	MQTYPE_STRING	Wartość pola `reply-to` we właściwościach AMQP, jeśli jest ustawiona.
AMQPCorrelationId	mq_amqp.Prp.Cid	MQTYPE_STRING	Wartość pola `correlation-id` we właściwościach AMQP, jeśli jest ustawiona jako łańcuch.

Tabela 108. Właściwości komunikatu AMQP (kontynuacja)			
Nazwa właściwości	**Nazwa MQRFH2**	**Typ**	**Opis**
		MQTYPE_BYTE_STRING	Wartość pola `correlation-id` we właściwościach AMQP, jeśli jest ustawiona jako łańcuch bajtów.
AMQPContentType	mq_amqp.Prp.Cnt	MQTYPE_STRING	Wartość pola `content-type` we właściwościach AMQP, jeśli jest ustawiona.
AMQPContentEncoding	mq_amqp.Prp.Cne	MQTYPE_STRING	Wartość pola `content-encoding` we właściwościach AMQP, jeśli jest ustawiona.
Czas AMQPAbsoluteExpiry	mq_amqp.Prp.Aet	MQTYPE_STRING	Wartość pola `absolute-expiry-time` we właściwościach AMQP, jeśli jest ustawiona.
AMQPCreationTime	mq_amqp.Prp.Crt	MQTYPE_STRING	Wartość pola `creation-time` we właściwościach AMQP, jeśli jest ustawiona.
AMQPGroupId	mq_amqp.Prp.Gid	MQTYPE_STRING	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	Wartość pola `reply-to-group-id` we właściwościach AMQP, jeśli jest ustawiona.

Każda z właściwości aplikacji z komunikatu AMQP jest ustawiana jako właściwość komunikatu produktu IBM MQ . Sekcję `application-properties` należy odtworzyć identycznie bajtowo-bajt-bajt, a więc mają zastosowanie następujące ograniczenia:

- Jeśli właściwość aplikacji zostanie odrzucona przez kod sprawdzania poprawności MQSETMP, komunikat zostanie odrzucony. Na przykład:
  - Długość nazwy właściwości jest ograniczona do wartości MQ_MAX_PROPERTY_NAME_LENGTH.
  - Nazwa właściwości musi być zgodna z regułami zdefiniowanymi przez specyfikację języka Java dla identyfikatorów Java.
  - Nazwa właściwości nie może zaczynać się od JMS lub `usr.`JMS , z wyjątkiem udokumentowanych właściwości JMS, które mogą być ustawione.
  - Nazwa właściwości nie może być słowem kluczowym SQL.
- Właściwość aplikacji zawierająca znak Unicode U+002E (".") powoduje, że komunikat zostanie odrzucony. Właściwość ta musi być wyrażona w grupie właściwości "usr" używanej przez usługę JMS.
- Obsługiwane są tylko właściwości: null, boolean, byte, short, int, long, float, double, binary i string. Właściwość aplikacji z dowolnym innym typem spowoduje, że komunikat zostanie odrzucony.

payload

- W przypadku elementu AMQP body z pojedynczą sekcją danych binarnych dane binarne (z wyjątkiem bitów AMQP) są umieszczane jako ładunek produktu IBM MQ z formatem MQFMT_NONE.

- W przypadku elementu AMQP body z pojedynczą sekcją danych łańcuchowych dane łańcuchowe (z wyjątkiem bitów AMQP) są umieszczane jako ładunek produktu IBM MQ , a format MQFMT_STRING jest formatowany.
- W przeciwnym razie program AMQP body tworzy ładunek w postaci formatu MQFMT_AMQP.

Zadania pokrewne

Tworzenie kanałów AMQP i korzystanie z nich

Zabezpieczanie klientów AMQP

ULW Niezawodność dostarczania komunikatów z AMQP

Istnieją cztery funkcje interfejsu API produktu IBM MQ , które umożliwiają sterowanie niezawodnością dostarczania komunikatów do aplikacji AMQP oraz z nich.

Są to:

- “Jakość usługi komunikatów (QOS)” na stronie 791
- “Automatyczne potwierdzenie subskrybenta” na stronie 792
- “Czas trwania subskrypcji” na stronie 792
- “Trwałość komunikatu” na stronie 792

Jakość usługi komunikatów (QOS)

Produkt MQ Light API oferuje dwie zalety usługi:

- Najwyżej raz
- Co najmniej raz

Użytkownik może wybrać jakość usługi, która 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 ramkę rozporządzania (dla subskrybentów) na wartość *true* lub *false*, w zależności od jakości usługi, która ma zostać zrealizowana.

Jakość usługi określa, kiedy komunikat jest odrzucany ze strony `sending` w konwersacji.

Publikowanie

Jeśli publikator wybierze opcję **QOS** *0* (co najwyżej jeden raz), publikator nie czeka na potwierdzenie z menedżera kolejek, zanim usunie jego kopię komunikatu.

Jeśli nawiązanie połączenia z menedżerem kolejek nie powiedzie się przed zakończeniem wysyłania, subskrybenty mogą nie być odbierane.

Jeśli publikator wybierze opcję **QOS** *1* (co najmniej raz), publikator oczekuje, aby menedżer kolejek potwierdził, że komunikat został zapisany do kolejek subskrybenta przed usunięciem jego kopii komunikatu.

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 opcję **QOS** *0* , menedżer kolejek nie będzie oczekiwać na potwierdzenie od subskrybenta przed usunięciem jego kopii komunikatu.

Jeśli połączenie z subskrybentem nie powiedzie się, zanim subskrybent odebrany zostanie komunikat, komunikat ten może zostać utracony.

Jeśli subskrybent wybierze opcję **QOS** *1* , menedżer kolejek oczekuje na potwierdzenie od subskrybenta, zanim usunie jego kopię komunikatu.

Jeśli połączenie z subskrybentem nie powiedzie się przed odebraniem komunikatu przez subskrybenta, komunikat jest przechowywany 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 przyjęcie każdego komunikatu, zanim menedżer kolejek odrzuci jego kopię. Subskrybent może zdecydować, kiedy będzie potwierdzać komunikaty.

Jeśli parametr **auto-confirm** ma wartość *true*, klient MQ Light automatycznie potwierdza dostarczenie każdego komunikatu po pomyślnym odebraniu przez klienta komunikatu przez sieć.

Zapewnia to, że jeśli wystąpi awaria sieci, komunikat zostanie ponownie dostarczony do aplikacji. Jednak aplikacja nie może utracić wiadomości, jeśli aplikacja nie powiedzie się między klientem MQ Light potwierdzającym komunikat, a aplikacją, która go przetwarza.

W przypadku opcji **auto-confirm** ustawionej na wartość *false*klient MQ Light nie potwierdza automatycznie dostarczenia wiadomości, ale pozostawia ją do aplikacji, aby zdecydować, kiedy powinna zostać potwierdzona.

Pozwala to aplikacji na dokonanie aktualizacji zasobu zewnętrznego, takiego jak baza danych lub plik, przed potwierdzeniem menedżera kolejek, że komunikat został przetworzony i można go usunąć.

Czas trwania subskrypcji

Gdy aplikacja subskrybuje, wybiera, czy subskrypcja i miejsce docelowe, w którym są przechowywane komunikaty dla tej subskrypcji, nadal istnieją po rozłączeniu aplikacji.

Opcja subskrypcji produktu MQ Light **ttl** służy do określania czasu (w milisekundach), przez jaki subskrypcja będzie nadal istnieć po rozłączonym aplikacji. Jeśli aplikacja ponownie nawiąże połączenie przed tym czasem, subskrypcja zostanie wznowiona, a aplikacja będzie mogła kontynuować korzystanie z komunikatów z tej subskrypcji.

Jeśli okres czasu życia jest przekazywany bez ponownego połączenia aplikacji, subskrypcja jest usuwana, a wszystkie komunikaty zapisane w miejscu docelowym są tracone, nawet jeśli są to komunikaty trwałe.

Jeśli ważne jest, aby nie tracić komunikatów, należy określić wartość czasu życia aplikacji, która jest wystarczająco wysoka, aby zapewnić, że komunikaty nie zostaną utracone w czasie wyłączenia.

Trwałość komunikatu

Trwałość komunikatów jest sterowana przez aplikacje publikującego i subskrybującego oraz konfigurację obiektów tematu produktu IBM MQ .

Jeśli subskrybent AMQP korzysta z produktu **QOS** *0* (co najwyżej raz) i utworzy nietrwałą subskrypcję, kanał AMQP zawsze umieszcza komunikaty nietrwałe w kolejce subskrybenta, niezależnie od innych opcji opisanych w poniższym tekście.

Należy pamiętać, że jeśli menedżer kolejek zostanie zatrzymany, a komunikaty zostaną utracone, a komunikaty zostaną utracone.

Jeśli publikator AMQP ustawia nagłówek AMQP **durable** na wartość *true*, kanał AMQP umieszcza trwałe komunikaty w kolejkach subskrybenta.

Jeśli menedżer kolejek zostanie zatrzymany z dowolnej przyczyny, komunikaty są nadal dostępne dla subskrybentów, gdy menedżer kolejek jest restartowany.

Jeśli nagłówek **durable** nie jest ustawiony, kanał AMQP wybiera trwałość opublikowanych komunikatów w oparciu o atrybut **DEFPSIST** odpowiedniego obiektu tematu IBM MQ .

Domyślnie jest to SYSTEM.BASE.TOPIC, w którym używany jest atrybut **DEFPSIST** o wartości *NO* (nietrwały).

Ostrzeżenie: Późniejsze wersje klienta MQ Light nie obsługują ustawiania trwałego nagłówka AMQP.

Zadania pokrewne

Tworzenie kanałów AMQP i korzystanie z nich

Zabezpieczanie klientów AMQP

ULW Topologie dla klientów AMQP z produktem IBM MQ

Przykładowe topologie, które ułatwiają programowanie klientów AMQP w pracy z produktem IBM MQ.

Zadania pokrewne

Tworzenie kanałów AMQP i korzystanie z nich

Zabezpieczanie klientów AMQP

ULW Klienty AMQP komunikujące się z produktem IBM MQ

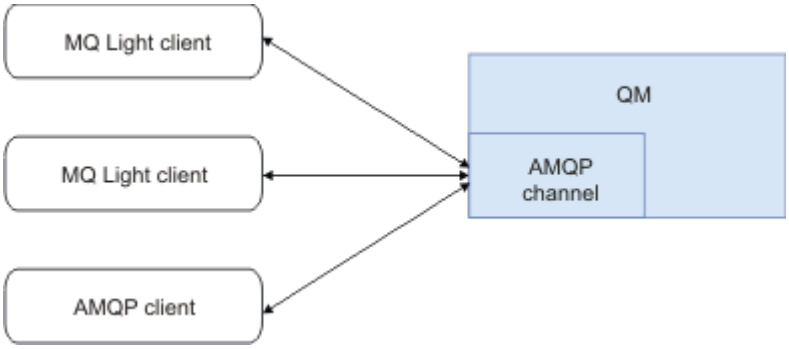
Produktu IBM MQ można używać jako dostawcy przesyłania komunikatów dla produktu IBM MQ Light lub dowolnej aplikacji, która jest zgodna z produktem AMQP 1.0. Mimo że każdy klient AMQP 1.0 może nawiązać połączenie z kanałem AMQP, niektóre funkcje AMQP nie są obsługiwane, na przykład transakcje lub wiele sesji.

Zdefiniowanie co najmniej jednego kanału AMQP powoduje, że klienty AMQP 1.0 mogą łączyć się z menedżerem kolejek i wysyłać komunikaty do łańcucha tematu. Klienci mogą również zasubskrybować wzorzec tematu, aby otrzymywać komunikaty zgodne z wzorcem.

W poniższym scenariuszu jedynymi aplikacjami, które wysyłają i odbierają komunikaty, są aplikacje MQ Light lub AMQP 1.0 .

Aplikacje mogą decydować, czy miejsca docelowe utworzone przez subskrybowanie łańcucha tematu są trwałe, dzięki czemu komunikaty nie zostaną utracone, jeśli aplikacja tymczasowo utraci połączenie z menedżerem kolejek.

Aplikacje mogą również wybrać, jak długo przechowywane są komunikaty przed wyczyszczniem z miejsca docelowego.



Zadania pokrewne

Tworzenie kanałów AMQP i korzystanie z nich

Zabezpieczanie klientów AMQP

ULW Klienty AMQP wymieniające komunikaty z aplikacjami produktu IBM MQ

Po zdefiniowaniu i uruchomieniu kanału AMQP aplikacje produktu MQ Light lub AMQP 1.0 mogą publikować komunikaty odbierane przez istniejące aplikacje produktu MQ . Komunikaty publikowane za pośrednictwem kanału AMQP są wysyłane do tematów produktu MQ , a nie do kolejek produktu MQ . Aplikacja MQ , która utworzyła subskrypcję przy użyciu wywołania API MQSUB, odbiera komunikaty opublikowane 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.

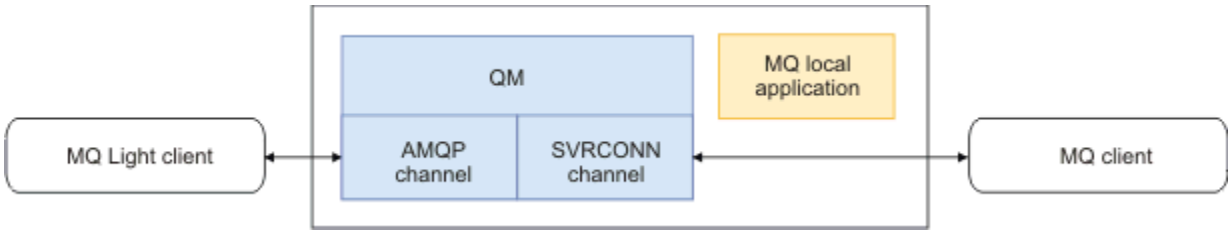
Dane komunikatu AMQP, atrybuty i właściwości są ustawiane w komunikacie MQ odebranym przez aplikację MQ . Więcej informacji na temat odwzorowań komunikatów AMQP na produkt MQ zawiera sekcja Odwzorowanie pól AMQP na pola produktu IBM MQ (komunikaty przychodzące).

Jeśli aplikacja MQ utworzyła subskrypcję, która jest trwała, komunikaty publikowane przez aplikację AMQP są zapisywane w kolejce, w której są tworzone kopie zapasowe subskrypcji. Komunikaty są następnie odbierane przez aplikację MQ , gdy aplikacja wznawia subskrypcję. Jeśli aplikacja AMQP określa czas komunikatu do życia, a aplikacja MQ nie nawiąże ponownie połączenia w czasie, w którym ma zostać nawiązane połączenie, komunikat ten utracił ważność z kolejki.

Aplikacje MQ Light lub AMQP 1.0 mogą również korzystać z komunikatów publikowanych przez istniejące aplikacje MQ . Komunikaty publikowane przez aplikacje produktu MQ w łańcuchach tematów lub tematów 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 typu time-to-live dla subskrypcji, a aplikacja AMQP rozłącza się przez dłuższy czas niż czas życia, subskrypcja utraciła ważność w menedżerze kolejek, a wszystkie komunikaty zapisane 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 AMQP znajduje się w sekcji Odwzorowanie pól AMQP na pola produktu IBM MQ (komunikaty wychodzące).



Zadania pokrewne

Tworzenie kanałów AMQP i korzystanie z nich

Zabezpieczanie klientów AMQP

ULW Konfigurowanie klientów AMQP do interakcji bezpośrednio z aplikacjami w kolejkach produktu IBM MQ

Implementacja produktu IBM MQ AMQP obsługuje tylko publikowanie/subskrybowanie. AMQP clients cannot put messages to, or get messages from, IBM MQ queues directly. Aby umożliwić publikowanie przez klienty AMQP komunikatów w istniejącej kolejce lub subskrybowanie klientów AMQP w celu odbierania komunikatów, które zostały umieszczone w kolejce, należy utworzyć dodatkowe obiekty.

Przegląd

Załóżmy na przykład, że istnieje aplikacja pobierający komunikaty z kolejki wejściowej `IN_QUEUE` i umieszczając te komunikaty w kolejce wyjściowej `OUT_QUEUE`. Klienty AMQP umożliwiają umieszczanie komunikatów w produkcie `IN_QUEUE`i pobieranie komunikatów z produktu `OUT_QUEUE` .

Uwaga: Do samej aplikacji nie są wymagane żadne zmiany.

Aby publikator AMQP umieł umieścić komunikat w kolejce, należy utworzyć subskrypcję administracyjną dla łańcucha tematu, do którego jest publikowana klient AMQP, z miejscem docelowym dla danej kolejki. Patrz “Wysyłanie komunikatów do aplikacji:” na stronie 795.

Aby subskrybent AMQP otrzymał komunikaty z kolejki, należy zastąpić kolejkę kolejką aliasową o tej samej nazwie, której celem jest obiekt tematu reprezentujący łańcuch tematu, do którego subskrybowany jest klient AMQP; patrz “Pobieranie komunikatów z aplikacji:” na stronie 795

Wysyłanie komunikatów do aplikacji:

Aplikacja już odbiera komunikaty z produktu `IN_QUEUE` , a użytkownik chce, aby klient AMQP mógł publikować komunikaty, tak aby przejść do tej kolejki, która będzie przetwarzana przez aplikację.

W tym celu należy utworzyć nową subskrypcję administracyjną, w której łańcuch tematu, z którego ta subskrypcja odbiera komunikaty, jest łańcuchem tematu publikowanego 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 `IN_QUEUE`.

Zakładając, że klient AMQP jest publikowany 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 opublikowane w programie `/application/in` są kierowane do miejsca docelowego `IN_QUEUE`, gdzie są one pobierane przez aplikację w taki sam sposób, jak inne komunikaty umieszczone w tej kolejce przez inne aplikacje.

Pobieranie komunikatów z aplikacji:

Aplikacja wprowadza komunikaty do produktu OUT_QUEUE, gdzie mogą być pobierane i przetwarzane przez innych klientów.

Jednak w tym przypadku komunikaty mają być dostarczane do klienta AMQP zamiast, ale klienty AMQP używają tylko publikowania/subskrypcji i nie mogą odbierać komunikatów bezpośrednio z kolejki.

Aby zastąpić klienty poprzednio odbierające komunikat przy użyciu klienta AMQP subskrybującego, należy utworzyć obiekt tematu dla łańcucha tematu, do którego jest zasubskrybowany klient AMQP, oraz kolejki aliasowej.

Ostrzeżenie: Jeśli użytkownik zdefiniuje kolejkę aliasową, a następnie uruchomi aplikację, zanim wszystkie klienty AMQP będą miały możliwość subskrybowania, komunikaty wysyłane przez aplikację wysyłają do "kolejki" (teraz temat) zostaną utracone, ponieważ nie ma subskrybentów.

Zmiany opisane w tym tekście zastępują klienty, które wcześniej odbierają komunikaty przy użyciu klienta AMQP subskrybującego tylko. Aby możliwe było korzystanie z kombinacji klientów AMQP i innych klientów w celu uzyskania komunikatów, konieczne jest wprowadzenie bardziej rozbudowanych zmian.

Zakładając, 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 dostarczone do tego obiektu tematu są dostarczane do klienta AMQP subskrybującego ten sam łańcuch tematu.

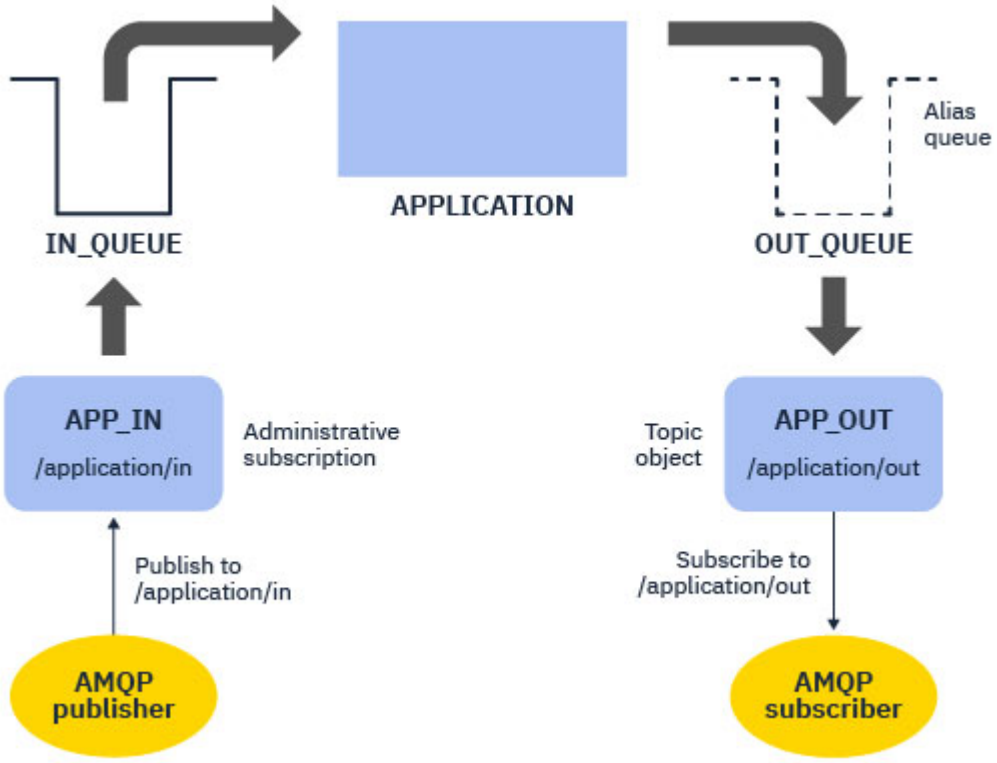
Następnie należy upewnić się, że komunikaty umieszczone w programie OUT_QUEUE przez aplikację są dostarczane do tego nowego obiektu tematu, tak aby były wysyłane do klienta subskrybującego.

W tym celu należy zastąpić istniejącą kolejkę OUT_QUEUE kolejką aliasową o tej samej nazwie, o typie docelowym utworzonego właśnie obiektu tematu, za pomocą następującej komendy MQSC:

```
DEF QALIAS(OUT_QUEUE) TARGTYPE(TOPIC) TARGET(APP_OUT)
```

Teraz komunikaty wprowadzone przez aplikację do programu OUT_QUEUE nie oczekują na odezwę kolejki. Zamiast tego są one dostarczane do miejsca docelowego tej kolejki aliasowej, to znaczy nowego obiektu tematu APP_OUT.

Klient AMQP, który jest subskrybowany w łańcuchu tematu reprezentowanym przez ten obiekt tematu `/application/out`, otrzymuje od kolejki aliasowej wszystkie komunikaty wysłane do tego obiektu tematu.



Zadania pokrewne

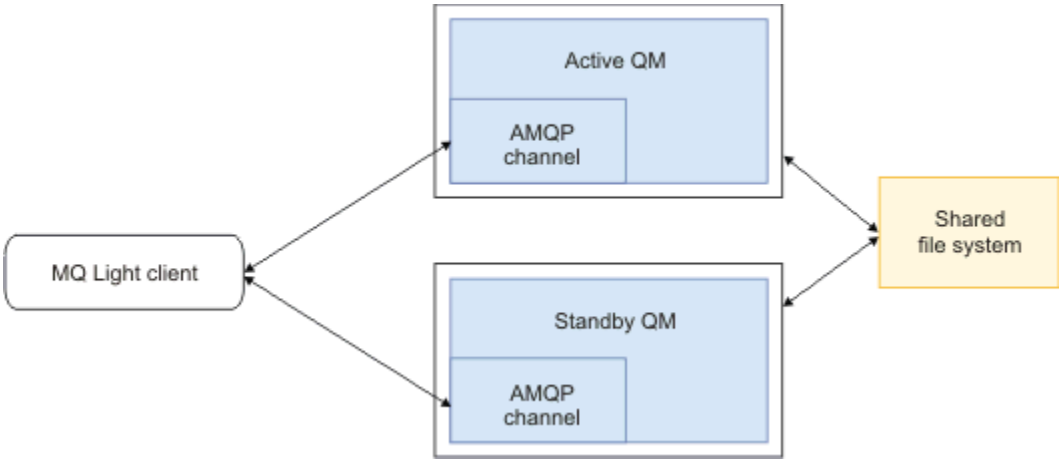
Tworzenie kanałów AMQP i korzystanie z nich

Zabezpieczanie klientów AMQP

ULW Konfigurowanie klienta AMQP w celu zapewnienia wysokiej dostępności

Istnieje możliwość skonfigurowania aplikacji MQ Light lub AMQP 1.0 w celu nawiązania połączenia z aktywną instancją menedżera kolejek z wieloma instancjami serwera IBM MQ i przełączenia awaryjnego do instancji rezerwowej menedżera kolejek z wieloma instancjami w parze wysokiej dostępności (HA). Aby to zrobić, należy skonfigurować aplikację AMQP z dwoma adresami IP i parami portów.

Interfejs API produktu MQ Light można skonfigurować przy użyciu funkcji niestandardowej, która jest wywoływana, jeśli klient utraci połączenie z serwerem. Funkcja może łączyć się z alternatywnym adresem IP, na przykład rezerwowym menedżerem kolejek produktu IBM MQ lub z oryginalnym adresem IP. W przypadku innych klientów AMQP, jeśli klient obsługuje konfigurację wielu punktów końcowych połączeń, należy skonfigurować aplikację z dwoma parami hosta i użyć funkcji ponownego połączenia udostępnianych przez bibliotekę AMQP w celu przełączenia się do rezerwowego menedżera kolejek.



Zadania pokrewne

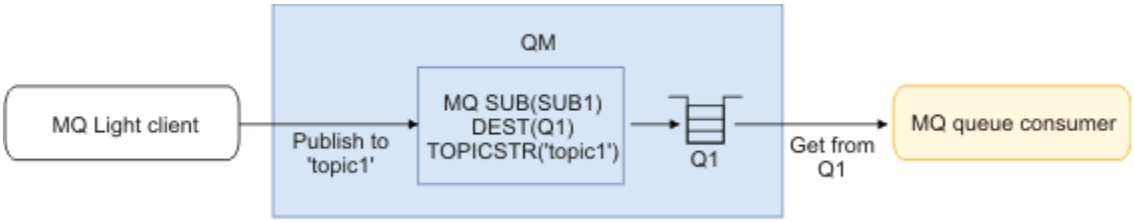
Tworzenie kanałów AMQP i korzystanie z nich

Zabezpieczanie klientów AMQP

ULW Konfigurowanie publikowania/subskrypcji dla klientów AMQP

Klienty AMQP mogą publikować w temacie przy użyciu subskrypcji produktu IBM MQ , która kieruje komunikaty do kolejki produktu IBM MQ odczytane przez istniejącą aplikację. Jeśli aplikacja MQ Light lub AMQP 1.0 ma wysyłać komunikaty do istniejącej aplikacji produktu IBM MQ skonfigurowanej do odczytu z kolejki, należy zdefiniować administrowaną subskrypcję programu IBM MQ w menedżerze kolejek.

Skonfiguruj subskrypcję tak, aby korzystała z wzorca tematu zgodnego z łańcuchem tematu używanym przez aplikację AMQP. Ustaw miejsce docelowe subskrypcji na nazwę kolejki, z której pobiera lub przegląda komunikaty aplikacji IBM MQ .



Zadania pokrewne

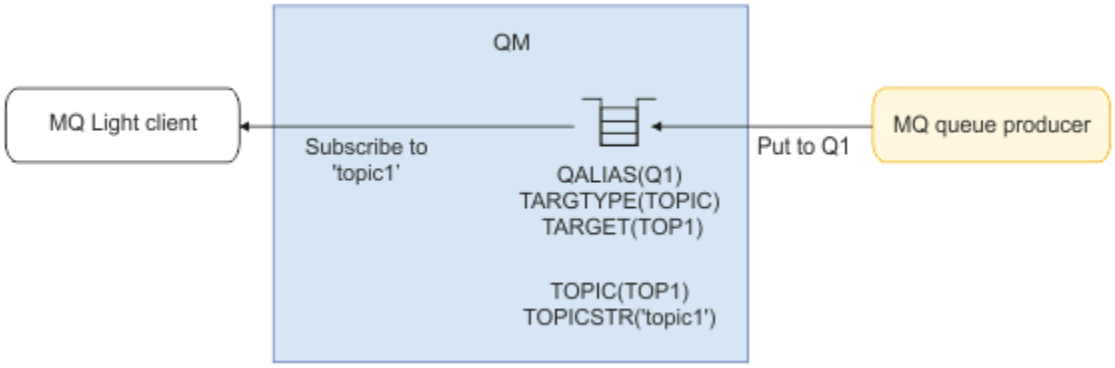
Tworzenie kanałów AMQP i korzystanie z nich

Zabezpieczanie klientów AMQP

ULW Klient AMQP używający aliasu kolejki do odbierania komunikatów z aplikacji IBM MQ

Klient AMQP może zasubskrybować temat i odbierać komunikaty umieszczone w kolejce aliasowej przez aplikację IBM MQ . Jeśli aplikacja MQ Light lub 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, która jest otwierana przez aplikację IBM MQ . Alias kolejki musi określać typ podstawowy TOPIC i obiekt podstawowy obiektu tematu IBM MQ , który ma łańcuch tematu zgodny z wzorcem tematu zasubskrybowanym przez aplikację AMQP.



Zadania pokrewne

Tworzenie kanałów AMQP i korzystanie z nich

Zabezpieczanie klientów AMQP

ULW Klient AMQP wysyła żądania do odpowiedzi i konsumuje odpowiedzi z serwera aplikacji

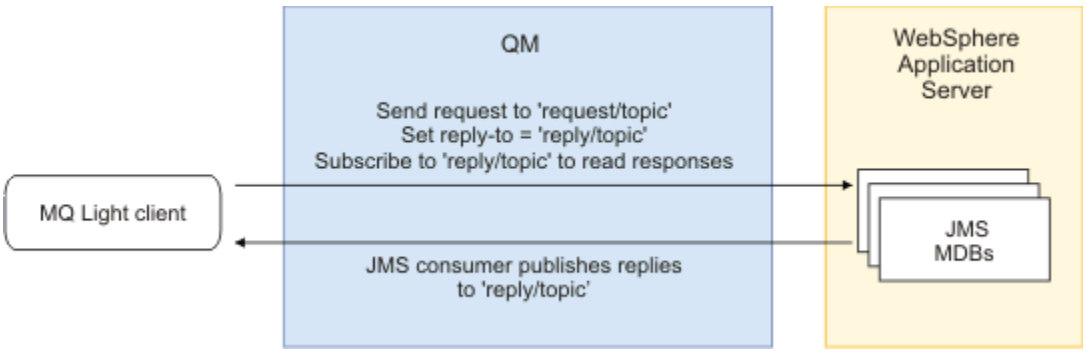
Klient MQ Light lub inny klient AMQP może wysyłać żądania do komponentu bean sterowanego komunikatami uruchomionego na serwerze aplikacji i konsumować odpowiedzi z tematu odpowiedzi. Produkt IBM MQ obsługuje aplikacje AMQP 1.0 służące do ustawiania tematu odpowiedzi w komunikatach publikowanych przez produkt IBM MQ . Gdy komunikat AMQP jest publikowany z zestawem atrybutów zwrotnych do atrybutu, wartość pola odpowiedzi jest ustawiana jako właściwość JMS dla konsumentów JMS, które mają być odbierane. To ustawienie umożliwia konsumentowi JMS odczytanie tematu odpowiedzi z komunikatu i wysłanie komunikatu odpowiedzi z powrotem do klienta AMQP.

Właściwość JMS to **JMSReplyTo**. Łańcuch odpowiedzi AMQP musi być jednym z następujących typów:

- Łańcuch tematu. Na przykład `'reply/topic'`
- Adres URL adresu AMQP w formularzu `amqp://host:port/[topic-string]`. Na przykład: `amqp://localhost:5672/reply/topic`

Jeśli jako pole odpowiedzi zostanie podany adres URL adresu AMQP, to przed ustawieniem właściwości **JMSReplyTo** zostanie usunięta cała reszta z wyjątkiem łańcucha tematu na końcu adresu URL.

Więcej informacji na temat odwzorowań z odpowiedzi na adres odpowiedzi AMQP na właściwość **JMSReplyTo** zawiera sekcja Odwzorowanie pól AMQP na pola produktu IBM MQ (komunikaty przychodzące) .



Zadania pokrewne

Tworzenie kanałów AMQP i korzystanie z nich

Zabezpieczanie klientów AMQP

ULW Wdrażanie aplikacji MQ Light w środowisku IBM MQ w siedzibie

Produkt IBM MQ obsługuje interfejs API przesyłania komunikatów produktu IBM MQ Light , dzięki czemu można użyć produktu IBM MQ do wdrożenia aplikacji MQ Light w lokalnym środowisku produktu IBM MQ .

Aplikacje produktu MQ Light można wdrażać w menedżerze kolejek produktu IBM MQ , umożliwiając aplikacjom produktu MQ Light komunikowanie się z istniejącymi aplikacjami korporacyjnymi, które są już połączone z produktem IBM MQ, tak jak pokazano na poniższym diagramie:

Aplikacje produktu MQ Light mogą współużytkować obszar tematu z istniejącymi aplikacjami produktu IBM MQ , co umożliwia im interakcję z istniejącymi systemami korporacyjnymi.

Zadania pokrewne

Tworzenie kanałów AMQP i korzystanie z nich

Zabezpieczanie klientów AMQP

V 9.1.0 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 publikować 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 REST API” na stronie 800. 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 (zCEE) to produkt z/OS , który umożliwia budowanie interfejsów REST API na podstawie istniejących zasobów aplikacyjnych systemu z/OS , takich jak transakcje CICS lub IMS , a także kolejek i tematów IBM MQ . Istniejący zasób aplikacyjny z/OS jest ukryty przed użytkownikiem. Dzięki temu można włączyć usługi REST, nie zmieniając ich ani żadnych istniejących aplikacji, które z nich korzystają.

Dzięki zCEE można łatwo zbudować REST API , który będzie wysyłać i odbierać dane JSON i opcjonalnie przekształcać te dane w bardziej tradycyjne struktury językowe oczekiwane przez wiele aplikacji mainframe. Na przykład struktury copybook języka COBOL.

Za pomocą edytora interfejsów API opartego na platformie Eclipse można zbudować obszerny interfejs API zgodny ze specyfikacją REST, korzystając z parametrów zapytania i segmentów ścieżki URL , manipulując formatem JSON przepływającym przez środowisko wykonawcze zCEE .

zCEE może być używane do udostępniania kolejek i tematów IBM MQ jako usług. Obsługiwane są dwa różne typy usług:

- Usługi jednokierunkowe: udostępniona funkcja jest podobna do tej, która jest udostępniana z mostem IBM MQ dla protokołu HTTP , w którym żądanie HTTP POST wysyła komunikat, a żądanie HTTP DELETE niszcząco pobiera komunikat. Główną korzyścią z zastosowania mostu jest wbudowana obsługa konwersji danych i możliwość użycia edytora interfejsów API do zbudowania bardziej wszechstronnego interfejsu API zgodnego ze specyfikacją REST.
- Usługi dwukierunkowe: udostępnia REST API na bazie pary kolejek używanych przez aplikację zaplecza typu żądanie-odpowiedź. Programy wywołujące wysyłają żądanie HTTP POST do usługi dwukierunkowej i wysyłają dane JSON. Te dane są umieszczane w kolejce żądań, w której są przetwarzane przez aplikację zaplecza, a odpowiedź jest umieszczana w kolejce odpowiedzi. Ta odpowiedź jest pobierana i wysyłana z powrotem do programu wywołującego jako treść odpowiedzi POST.

Produkt zCEE jest obsługiwany w systemie IBM MQ 8 i nowszych. Więcej informacji na ten temat zawiera sekcja IBM MQ for z/OS Service Provider for z/OS Connect.

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.

V 9.1.0 Przesyłanie komunikatów przy użyciu REST API

Za pomocą programu messaging REST API można wykonywać proste przesyłanie komunikatów typu punkt z punktem i publikowanie . Użytkownik może publikować komunikaty w temacie, wysyłać komunikaty do kolejki, przeglądać komunikaty w kolejce, oraz destrukcyjnie pobrać komunikaty z kolejki. Informacje są wysyłane do messaging REST API i odbierane z niego w formacie zwykłego tekstu.

Zanim rozpoczniesz

Uwaga:

- Opcja messaging REST API jest domyślnie włączona. Aby zapobiec wszystkim przesyłaniu komunikatów, można wyłączyć messaging REST API . Więcej informacji na temat włączania i wyłączania konsoli messaging REST APIzawiera sekcja Konfigurowanie produktu messaging REST API.
- Produkt messaging REST API jest zintegrowany z zabezpieczeniami produktu IBM MQ . Aby można było używać produktu messaging REST API, użytkownicy muszą być uwierzytelniani na serwerze mqweb i muszą należeć do roli MQWebUser . Użytkownik musi być również uprawniony do uzyskania dostępu do określonej kolejkilub tematu. Więcej informacji na temat zabezpieczeń dla produktu REST API zawiera sekcja Zabezpieczenia konsoli IBM MQ i REST API.
- Jeśli produkt Advanced Message Security (AMS) jest używany z produktem messaging REST API, należy pamiętać, że wszystkie komunikaty są szyfrowane przy użyciu kontekstu serwera mqweb, a nie kontekstu użytkownika, który publikuje ten komunikat.
- Podczas odbierania komunikatów lub przeglądania obsługiwane są tylko komunikaty w formacie IBM MQ MQSTR . Następnie wszystkie komunikaty są destruktywnie odbierane w punkcie synchronizacji, a wszystkie nieobsługiwane komunikaty są pozostawiane w kolejce. Kolejka IBM MQ może zostać skonfigurowana w celu przeniesienia tych komunikatów nieprzetwarzalnych do alternatywnego miejsca docelowego. Więcej informacji zawiera sekcja “Obsługa komunikatów nieprzetwarzalnych w produkcie IBM MQ classes for JMS” na stronie 223.
- messaging REST API nie daje jednorazowej i jednorazowej operacji dostarczania wiadomości z obsługą transakcyjną. Jeśli zostanie wywołana metoda HTTP POST, a połączenie nie powiedzie się, zanim klient odbierze odpowiedź HTTP, klient nie może od razu stwierdzić, czy komunikat został wysłany do określonej kolejkilub opublikowany w określonym temacie. Jeśli zostanie wydane żądanie HTTP DELETE, a połączenie nie powiedzie się przed odebraniem odpowiedzi HTTP przez klienta, komunikat mógł zostać zniszczony z kolejki i utracony, ponieważ nie ma możliwości wycofywania destrukcyjnego powrotu.
- Nowe wiersze w przychodzących łańcuchach są usuwane z operacji POST HTTP. Aplikacje REST

  nie należy używać nowych wierszy w komunikatach wysyłanych lub opublikowanych przy użyciu interfejsu API usług REST, ponieważ zostaną one utracone.

Procedura

- “Pierwsze kroki w produkcie messaging REST API” na stronie 801
- “Korzystanie z messaging REST API” na stronie 804
- Obsługa błędów systemuREST API
- WykrywanieREST API
- Obsługa języków narodowych w produkcieREST API

Odsyłacze pokrewne

Przesyłanie komunikatów REST API -odwołanie

Informacje pokrewne

Kurs: Pierwsze kroki z przesyłaniem komunikatów programu IBM MQ REST API

V 9.1.0 Pierwsze kroki w produkcie messaging REST API

Szybkie rozpoczęcie pracy z programem messaging REST API i wypróbowanie kilku przykładowych komend za pomocą komendy cURL.

Zanim rozpoczniesz

Aby rozpocząć korzystanie z produktu messaging REST API, przykłady w tej czynności mają następujące wymagania:

- W przykładach cURL jest używany do wysyłania żądań REST w celu umieszczenia komunikatów w kolejce i pobierania komunikatów z kolejki. Dlatego aby wykonać to zadanie, należy zainstalować w systemie cURL .
- W przykładach używany jest menedżer kolejek QM1. Utwórz menedżer kolejek o tej 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 systemie IBM ikomendy powinny być uruchomione w systemie QSHELL.

Procedura

1. Upewnij się, że serwer mqweb jest skonfigurowany dla serwera messaging REST API:
   - Jeśli serwer mqweb nie jest jeszcze skonfigurowany do użycia przez produkt administrative REST API, produkt administrative REST API for MFT, produkt messaging REST APIlub produkt IBM MQ Console, skonfiguruj serwer mqweb.Więcej informacji na temat tworzenia podstawowej konfiguracji serwera mqweb z podstawowym rejestrem zawiera sekcja Podstawowa konfiguracja serwera mqweb.
   - Jeśli serwer mqweb jest już skonfigurowany, upewnij się, że dodano odpowiednich użytkowników, aby umożliwić przesyłanie komunikatów w kroku 5.
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 produktu REST API URL , wprowadzając następującą komendę:

   ```
   dspmqweb status
   ```

   W przykładach przedstawionych w poniższych krokach przyjęto, że REST API URL jest domyślnym URL https://localhost:9443/ibmmq/rest/v1/. Jeśli używany jest adres URL inny niż domyślny, należy podać go w poniższych krokach.
4. Utwórz kolejkę MSGQw menedżerze kolejek QM1. Ta kolejka jest używana do przesyłania komunikatów. Zastosuj jedną z następujących metod:
   - Użyj żądania POST dla zasobu queue w administrative REST API, uwierzytelniając jako użytkownik mqadmin :

     ```
     curl -k https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1/queue -X POST -u
     mqadmin:mqadmin -H "ibm-mq-rest-csrf-token: value" -H "Content-Type: application/json" --
     data "{\"name\":\"MSGQ\"}"
     ```

   - Użyj komend MQSC:

z/OS W systemie z/OSzamiast komendy **runmqsc** należy użyć źródła 2CR . Więcej informacji na ten temat zawiera sekcja Źródła, z których można wydawać komendy MQSC w systemie z/OS.

a. Uruchom program **runmqsc** dla menedżera kolejek, wprowadzając następującą komendę:

```
runmqsc QM1
```

b. Użyj komendy MQSC **DEFINE QLOCAL** , aby utworzyć kolejkę:

```
DEFINE QLOCAL(MSGQ)
```

c. Wyjdź z programu **runmqsc** , wprowadzając następującą komendę:

```
end
```

5. Nadaj uprawnienia użytkownikowi, który został dodany do `mqwebuser.xml` w kroku 5 sekcji Konfiguracja podstawowa dla serwera mqweb , aby uzyskać dostęp do kolejki `MSGQ`. Zastąp użytkownika, w którym używany jest produkt `myuser` :

- z/OS W 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 dostęp do identyfikatora użytkownika uruchomionego zadania mqweb, aby ustawić cały kontekst w kolejce:

  ```
  RDEFINE MQADMIN hlq.CONTEXT.MSGQ UACC(NONE)
  PERMIT hlq.CONTEXT.MSGQ CLASS(MQADMIN) ID(mqwebStartedTaskID) ACCESS(CONTROL)
  ```

- Multi W przypadku wszystkich innych systemów operacyjnych, jeśli użytkownik znajduje się w grupie mqm, uprawnienia są już nadawane. 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 przekazać użytkownikowi przeglądanie, sprawdzanie, pobieranie i umieszczanie uprawnień do kolejki:

  ```
  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`, korzystając z żądania POST w zasobie `message` . Substitute your user ID and password from the `mqwebuser.xml` for `myuser` and `mypassword`:

   Używane jest uwierzytelnianie podstawowe, a nagłówek HTTP produktu `ibm-mq-rest-csrf-token` 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/v1/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. Destruktywnie pobierz komunikat z kolejki `Hello World!` w kolejce `MSGQ` w menedżerze kolejek `QM1`, korzystając z żądania DELETE w zasobie `message` . Substitute your user ID and password from the `mqwebuser.xml` for `myuser` and `mypassword`:

```
curl -k https://localhost:9443/ibmmq/rest/v1/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 korzystania z produktu messaging REST API i konstruowania adresów URL z parametrami zapytania: “Korzystanie z messaging REST API” na stronie 804.
- V 9.1.2 Jeśli używany jest produkt messaging REST API, połączenia z menedżerem kolejek są łączone w celu zoptymalizowania wydajności. Można skonfigurować maksymalną wielkość puli oraz jakie działanie jest podejmowane, gdy wszystkie połączenia w puli będą używane: Konfigurowanie produktu messaging REST API.
- Przejrzyj informacje uzupełniające dotyczące dostępnych zasobów produktu messaging REST API i wszystkich dostępnych parametrów zapytania: messaging REST API reference.
- Wykryj administrative REST API, interfejs RESTful do administrowania produktem IBM MQ : Administrowanie za pomocą REST API.
- Wykryj IBM MQ Console, oparty na przeglądarce interfejs GUI: Administrowanie za pomocą IBM MQ Console.

V 9.1.0 Korzystanie z messaging REST API

Podczas korzystania z produktu messaging REST APImożna wywoływać metody HTTP w adresach URL w celu wysyłania i odbierania komunikatów produktu IBM MQ . Metoda HTTP, na przykład POST, reprezentuje typ działania, które ma zostać wykonane dla obiektu, który jest reprezentowany przez adres URL. Dalsze informacje na temat działania mogą być zakodowane w parametrach zapytania. Informacje na temat wyniku wykonania działania mogą zostać zwrócone jako treść odpowiedzi HTTP.

Zanim rozpoczniesz

Przed użyciem produktu messaging REST APInależy rozważyć następujące kwestie:

- Aby można było korzystać z produktu messaging REST API, należy uwierzytelnić się na serwerze mqweb. Uwierzytelnianie można uwierzytelnić za pomocą podstawowego uwierzytelniania HTTP, uwierzytelniania certyfikatu klienta lub uwierzytelniania opartego na tokenie. Więcej informacji na temat korzystania z tych metod uwierzytelniania zawiera sekcja Zabezpieczenia konsoliIBM MQ i zabezpieczenia produktu REST API.
- W programie REST API jest rozróżniana wielkość liter. Na przykład żądanie HTTP POST na poniższym adresie URL powoduje wystąpienie błędu, jeśli menedżer kolejek ma nazwę `qmgr1`.

```
/ibmmq/rest/v2/messaging/qmgr/QMGR1/queue/Q1/message
```

- Nie wszystkie znaki, które mogą być używane w nazwach obiektów IBM MQ , mogą być bezpośrednio zakodowane w adresie URL. Aby zakodować te znaki poprawnie, należy użyć odpowiedniego kodowania adresu URL:
  - Ukośnik musi być zakodowany jako %2F.
  - Znak procentu musi być zakodowany jako %25.

  - Okres musi być zakodowany jako %2E.
  - Znak zapytania musi być zakodowany jako %3F.
- Podczas odbierania komunikatów lub przeglądania obsługiwane są tylko komunikaty w formacie IBM MQ MQSTR . Następnie wszystkie komunikaty są destruktywnie odbierane w punkcie synchronizacji, a wszystkie nieobsługiwane komunikaty są pozostawiane w kolejce. Kolejka IBM MQ może zostać skonfigurowana w celu przeniesienia tych komunikatów nieprzetwarzalnych do alternatywnego miejsca docelowego. Więcej informacji zawiera sekcja "Obsługa komunikatów nieprzetwarzalnych w produkcie IBM MQ classes for JMS" na stronie 223.

O tym zadaniu

Jeśli do wykonywania działań związanych z przesyłaniem komunikatów w obiekcie kolejki produktu IBM MQ jest używany produkt REST API , należy najpierw utworzyć adres URL reprezentujący ten obiekt. Każdy adres URL rozpoczyna się od przedrostka, który opisuje nazwę hosta i port, do którego ma zostać wysłane żądanie. Pozostała część adresu URL opisuje konkretny obiekt lub trasę do tego obiektu, znaną jako zasób.

Działanie przesyłania komunikatów, które ma zostać wykonane na zasobie, określa, czy adres URL wymaga parametrów zapytania. Definiuje także używaną metodę HTTP oraz informację o tym, czy do adresu URL są wysyłane dodatkowe informacje, czy też są one zwracane z tego adresu. Dodatkowe informacje mogą stanowić część żądania HTTP lub być zwracane jako część odpowiedzi HTTP.

Po utworzeniu adresu URL można wysłać żądanie HTTP do produktu IBM MQ. Żądanie można wysłać, korzystając z implementacji HTTP wbudowanej w wybrany język programowania. Żądanie można również wysłać za pomocą narzędzi wiersza komend, takich jak cURL, przeglądarki WWW lub dodania przeglądarki WWW.

Ważne: Należy wykonać co najmniej kroki "1.a" na stronie 805 i "1.b" na stronie 805.

Procedura

1. Utwórz adres URL:
   a) Określ adres URL przedrostka, wprowadzając następującą komendę:

   ```
   dspmqweb status
   ```

   Adres URL, który ma być używany, zawiera frazę `/ibmmq/rest/` .

   b) Dodaj kolejkę i powiązane zasoby menedżera kolejek, które mają być używane na potrzeby przesyłania komunikatów do ścieżki adresu URL.

   W odwołaniu do przesyłania komunikatów segmenty zmiennych można zidentyfikować w adresie URL za pomocą nawiasów klamrowych otaczających ją *{ }*. Więcej informacji zawiera temat / messaging/qmgr/{qmgrName}/queue/{queueName}/message.

   Na przykład, aby umożliwić interakcję z kolejką *Q1* powiązaną z menedżerem kolejek *QM1*, należy dodać `/qmgr` i `/queue` do przedrostka adresu URL, aby utworzyć następujący adres URL:

   ```
   https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/Q1/message
   ```

   c) Opcjonalne: Dodaj opcjonalny parametr zapytania do adresu URL.

   Czy dodać znak zapytania?, Parametr zapytania, znak równości =, a wartość adresu URL.

   Aby na przykład poczekać maksymalnie 30 sekund, aby następny komunikat stał się dostępny, należy utworzyć następujący adres URL:

   ```
   https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/Q1/message?wait=30000
   ```

   d) Opcjonalne: Dodaj kolejne opcjonalne parametry zapytania do adresu URL.

   Dodaj znak ampersand &, do adresu URL, a następnie powtórz krok 1c.

2. Wywołaj odpowiednią metodę HTTP w adresie URL. Określ dowolny opcjonalny ładunek komunikatu i podaj odpowiednie referencje zabezpieczeń do uwierzytelnienia. Na przykład:
   - Użyj implementacji HTTP/REST wybranego języka programowania.
   - Należy użyć narzędzia, takiego jak program dodatkowy przeglądarki klienta REST lub produkt cURL.

Tworzenie aplikacji MQI za pomocą programu IBM MQ

Produkt IBM MQ udostępnia 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 znajduje się w sekcji Opisy wywołań. Ten temat zawiera listę wywołań MQI, a każdy z wywołań pokazuje, jak należy zakodować wywołania w każdym z tych języków.

Produkt IBM MQ udostępnia pliki definicji danych, które ułatwiają pisanie aplikacji. Pełny opis znajduje się w sekcji “Pliki definicji danych produktu IBM MQ” na stronie 806.

W celu ułatwienia wyboru języka proceduralnego w celu kodowania programów należy wziąć pod uwagę maksymalną długość komunikatów, które będą przetwarzane przez programy. Jeśli programy będą przetwarzać tylko komunikaty o znanej maksymalnej długości, można zakodować je w dowolnym obsługiwanym języku. Jeśli maksymalna długość komunikatów, które programy będą musiały być przetwarzane, nie jest znakowa, język wybrany przez użytkownika będzie zależał od tego, czy używany jest program CICS, IMS, czy też aplikacja wsadowa:

IMS i wsadowe
Kod programów w języku C, PL/I lub asembler języka, aby korzystać z udogodnień tych języków oferują do uzyskania i zwolnienia arbitralne ilości pamięci. Alternatywnie można zakodować programy w języku COBOL, ale używać języków asemblera, języka PL/I lub C, aby uzyskać i zwolnić pamięć masową.

CICS
Kod programu w dowolnym języku obsługiwanym przez produkt CICS. Interfejs EXEC CICS udostępnia wywołania służące do zarządzania pamięcią, jeśli jest to konieczne.

Pojęcia pokrewne

“Aplikacje obiektowe” na stronie 14
Produkt IBM MQ zapewnia obsługę elementów JMS, Java, C + +, .NETi ActiveX. Te języki i środowiska używają modelu obiektów IBM MQ , który udostępnia klasy udostępniające te same funkcje, co wywołania i struktury produktu IBM MQ .

Przegląd techniczny

“Pojęcia związane z projektowaniem aplikacji” na stronie 7
Do pisania aplikacji IBM MQ można użyć wyboru języków proceduralnych lub obiektowych. Przed rozpoczęciem projektowania i pisania aplikacji produktu IBM MQ należy zapoznać się z podstawowymi pojęciami dotyczącymi produktu IBM MQ .

Odsyłacze pokrewne

Informacje dodatkowe dotyczące programowania aplikacji

Pliki definicji danych produktu IBM MQ

Produkt IBM MQ udostępnia pliki definicji danych, które ułatwiają pisanie aplikacji.

Pliki definicji danych są również znane jako:

Język	Definicje danych
C	Włącz pliki lub pliki nagłówkowe
Visual Basic	Pliki modułów (tylko wersje 32-bitowe)

Język	Definicje danych
COBOL	Kopiuj pliki
Asembler	Makra
PL/I	Pliki INCLUDE

Pliki definicji danych ułatwiające pisanie wyjść kanału są opisane w sekcji IBM MQ COPY, header, include, and module files.

Pliki definicji danych, które ułatwiają pisanie programów zewnętrznych, są opisane w sekcji “Procedury zewnętrzne, wyjścia funkcji API i usługi instalowalne produktu IBM MQ” na stronie 1033.

Pliki definicji danych obsługiwane w języku C + + znajdują się w sekcji Korzystanie z języka C++.

IBM i

Pliki definicji danych obsługiwane w języku RPG można znaleźć w podręczniku IBM i Application Programming Reference (ILE/RPG).

Nazwy plików definicji danych mają przedrostek CMQ, a także przyrostek określony przez język programowania:

Przyrostek	Język
a	Język asembler
b	Visual Basic
c	C
l	COBOL (bez wartości inicjalizowanych)
p	PL/I
v	COBOL (z ustawionym domyślnym zestawem wartości)

Biblioteka instalacji

Nazwa **thlqual** jest kwalifikatorem wysokiego poziomu dla biblioteki instalacji w systemie z/OS.

W tej sekcji przedstawiono pliki definicji danych produktu IBM MQ , które są dostępne pod następującymi nagłówkami:

- “Pliki włączanych języków C” na stronie 807
- “Pliki modułu Visual Basic” na stronie 808
- “Pliki kopii COBOL” na stronie 808
- z/OS “Asembler System/390 -makra języka” na stronie 809
- z/OS “Pliki włączanych PL/I” na stronie 809

Pliki włączanych języków C

Pliki włączanych aplikacji IBM MQ C są wymienione w sekcji Pliki nagłówkowe języka C. Są one instalowane w następujących katalogach lub bibliotekach:

Platforma	Katalog instalacyjny lub biblioteka
IBM i IBM i	QMQM/H
UNIX UNIX	*MQ_INSTALLATION_PATH*/inc/

Platforma	Katalog instalacyjny lub biblioteka
Windows Systemy Windows	*MQ_INSTALLATION_PATH*\Tools\c\include
z/OS z/OS	**thlqual**.SCSQC370

gdzie *MQ_INSTALLATION_PATH* reprezentuje katalog najwyższego poziomu, w którym zainstalowano produkt IBM MQ .

Uwaga: W przypadku produktu UNIXpliki włączanych plików są dowiązane symbolicznie do produktu `/usr/include`.

Więcej informacji na temat struktury katalogów zawiera sekcja Planowanie obsługi systemu plików.

Pliki modułu Visual Basic

Produkt IBM MQ for Windows udostępnia cztery pliki modułu Visual Basic.

Są one wymienione w sekcji Pliki modułu Visual Basic i są zainstalowane 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 oraz dwa pliki kopii dla każdej z tych struktur.

Istnieją dwa pliki kopii dla każdej struktury, ponieważ każda z nich jest udostępniana zarówno z wartościami początkownymi, jak i bez nich:

- W sekcji WORKING-STORAGE SECTION w programie w języku COBOL należy użyć plików, które inicjują pola struktury, do wartości domyślnych. Struktury te są zdefiniowane w plikach kopii, których nazwy są przyrostowe z literą V (wartości).
- W sekcji LINKAGE w programie w języku COBOL należy używać struktur bez wartości początkowych. Struktury te są definiowane w plikach kopii, których nazwy są przyrostowe z literą L (powiązanie).

IBM i Pliki kopii zawierające definicje danych i interfejsu dla produktu IBM i są udostępniane dla programów w języku ILE COBOL za pomocą wywołań prototypowych dla interfejsu MQI. Pliki istnieją w QMQM/QCBLLESRC z nazwami elementów, które mają przyrostek L (dla struktur bez wartości początkowych) lub przyrostkiem V (dla struktur o wartościach początkowych).

Pliki kopii w języku COBOL produktu IBM MQ są wymienione w sekcji Pliki kopii COBOL. Są one instalowane w następujących katalogach:

Platforma	Katalog instalacyjny lub biblioteka
UNIX Inne platformy UNIX	*MQ_INSTALLATION_PATH*/inc/
IBM i IBM i	QMQM/QCBLLESRC
Windows Windows	*MQ_INSTALLATION_PATH*\Tools\cobol\copybook (for Micro Focus COBOL) *MQ_INSTALLATION_PATH*\Tools\cobol\copybook\VAcobol (w przypadku produktu IBM VisualAge COBOL)
z/OS z/OS	**thlqual**.SCSQCOBC

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Dołącz do programu tylko te pliki, które są potrzebne. Należy to zrobić z jedną lub większą liczbę instrukcji COPY po deklaracji level-01 . Oznacza to, że w razie potrzeby można dołączyć wiele wersji struktur w programie. Należy zauważyć, że CMQV jest dużym plikiem.

Poniżej przedstawiono przykład kodu COBOL w celu uwzględnienia pliku 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 , po której następuje instrukcja COPY, która ma zostać skopiowana do pozostałej części deklaracji struktury. Aby odwołać się do odpowiedniej instancji, należy użyć słowa kluczowego IN.

Poniżej przedstawiono przykład kodu COBOL w celu uwzględnienia dwóch instancji 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 w granicach 4-bajtowych. W przypadku użycia instrukcji COPY w celu uwzględnienia struktury następującej po elemencie, który nie jest elementem level-01 , należy upewnić 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 przedstawiają 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 wstępnie ustalone w ten sposób.

Nazwy pól w deklaracjach w plikach kopii struktury są pisane wielkimi literami. Zamiast tego można użyć małych liter lub małych liter. Na przykład pole *StrucId* struktury MQGMO jest wyświetlane jako MQGMO-STRUCID w deklaracji języka COBOL i w pliku kopii.

Struktury przyrostków V są deklarowane przy użyciu wartości początkowych dla wszystkich pól, dlatego należy ustawić tylko te pola, w których wymagana wartość jest inna od wartości początkowej.

Asembler System/390 -makra języka

z/OS

Produkt IBM MQ for z/OS udostępnia dwa makra języka asemblera zawierające nazwane stałe i jedno makro do generowania każdej struktury.

Są one wymienione w pliku z/OS Assembler COPY files i są instalowane w pliku **thlqual**.SCSQMACS.

Makra te są wywoływane za pomocą kodu w następujący sposób:

```
MY_MQMD CMQMDA EXPIRY=0,MSGTYPE=MQMT_DATAGRAM
```

Pliki włączanych PL/I

z/OS

Program IBM MQ for z/OS zawiera pliki zawierające wszystkie definicje, które są potrzebne podczas pisania aplikacji IBM MQ w PL/I.

Pliki te są wymienione w katalogu PL/I include files i są instalowane w katalogu **thlqual**.SCSQPLIC:

Uwzględnij te pliki w programie, jeśli kod pośredniczący produktu IBM MQ ma zostać dowiązany do programu (patrz “Przygotowanie programu do uruchomienia” na stronie 1129). Uwzględnij tylko produkt CMQP, jeśli planowane jest połączenie dynamicznie wywołań produktu IBM MQ (patrz “Dynamiczne wywoływanie kodu pośredniczącego produktu IBM MQ” na stronie 1136). Łączenie dynamiczne może być wykonywane tylko dla programów wsadowych i IMS .

Pisanie aplikacji proceduralnej w celu kolejkowania

Ta sekcja zawiera informacje na temat pisania aplikacji kolejkowania, łączenia się i rozłączania z menedżerem kolejek, publikowania/subskrypcji oraz obiektów otwierających i zamykających.

Aby dowiedzieć się więcej na temat pisania aplikacji, skorzystaj z poniższych odsyłaczy:

- “Interfejs kolejki komunikatów-przegląd” na stronie 811
- “Nawiąże połączenie z menedżerem kolejek i odłączenie go od niego” na stronie 824
- “Otwieranie i zamykanie obiektów” na stronie 833
- “Umieszczanie komunikatów w kolejce” na stronie 844
- “Pobieranie komunikatów z kolejki” na stronie 860
- “Zapisywanie aplikacji publikowania/subskrypcji” na stronie 901
- “Sprawdzanie i ustawianie atrybutów obiektu” na stronie 943
- “Zatwierdzanie i wycofywanie jednostek pracy” na stronie 946
- “Uruchamianie aplikacji produktu IBM MQ przy użyciu wyzwalaczy” na stronie 958
- “Praca z interfejsem MQI i klastrami” na stronie 978
- z/OS “Używanie i zapisywanie aplikacji w systemie IBM MQ for z/OS” na stronie 983
- z/OS “Aplikacje pomostowe IMS i IMS w systemie IBM MQ for z/OS” na stronie 70

Pojęcia pokrewne

“Pojęcia związane z projektowaniem aplikacji” na stronie 7
Do pisania aplikacji IBM MQ można użyć wyboru języków proceduralnych lub obiektowych. Przed rozpoczęciem projektowania i pisania aplikacji produktu IBM MQ należy zapoznać się z podstawowymi pojęciami dotyczącymi produktu IBM MQ .

“Tworzenie aplikacji dla składnika IBM MQ” na stronie 5
Istnieje możliwość tworzenia aplikacji w celu wysyłania i odbierania komunikatów oraz do zarządzania menedżerami kolejek i powiązanymi zasobami. Produkt IBM MQ obsługuje aplikacje napisane w wielu różnych językach i w różnych ramach.

“Uwagi dotyczące projektowania aplikacji produktu IBM MQ” na stronie 50
Po zdecydowaniu, w jaki sposób aplikacje mogą korzystać z platform i środowisk, które są dostępne dla użytkownika, należy zdecydować, w jaki sposób korzystać z funkcji oferowanych przez produkt IBM MQ.

“Pisanie aplikacji proceduralnych klienta” na stronie 1008
Co należy wiedzieć, aby pisać aplikacje klienckie w systemie IBM MQ , korzystając z języka proceduralnego.

“Budowanie aplikacji proceduralnej” na stronie 1099
Aplikację IBM MQ można napisać w jednym z kilku języków proceduralnych, a następnie uruchomić aplikację na kilku różnych platformach.

“Obsługa proceduralnych błędów programu” na stronie 1145
Te informacje wyjaśniają błędy związane z wywołaniami MQI aplikacji, gdy wywołuje połączenie, lub gdy jego komunikat jest dostarczany do miejsca docelowego.

Zadania pokrewne

“Korzystanie z przykładowych programów proceduralnych produktu IBM MQ” na stronie 1165
Te przykładowe programy są zapisywane w językach proceduralnych i demonstrują typowe zastosowania interfejsu kolejki komunikatów (Message Queue Interface-MQI). Programy IBM MQ na różnych platformach.

Interfejs kolejki komunikatów-przegląd

Informacje na temat komponentów interfejsu kolejek komunikatów (Message Queue Interface-MQI).

Interfejs kolejki komunikatów składa się z następujących elementów:

- *Wywołania* , za pomocą których programy mogą uzyskiwać dostęp do menedżera kolejek i jego obiektów
- *Struktury* , których programy używają do przekazywania danych do menedżera kolejek i pobierania danych z niego
- *Elementarne typy danych* służące do przekazywania danych do menedżera kolejek i pobierania danych z tego menedżera kolejek.

z/OS IBM MQ for z/OS również:

- Dwa dodatkowe wywołania, za pośrednictwem których programy wsadowe programu z/OS mogą zatwierdzać zmiany i wycofać ich zmiany.
- *Pliki definicji danych* (czasami znane jako pliki kopii, makra, pliki nagłówkowe i pliki nagłówkowe), które definiują wartości stałych dostarczanych wraz z produktem IBM MQ for z/OS.
- *Programy pośredniczenia* w celu połączenia edycji z aplikacjami.
- Pakiet przykładowych programów demonstrujący sposób korzystania z interfejsu MQI na platformie z/OS . Więcej informacji na temat tych przykładów można znaleźć w sekcji “Korzystanie z przykładowych programów dla produktu z/OS” na stronie 1272.

IBM i IBM MQ for IBM i również:

- *Pliki definicji danych* (czasami znane jako pliki kopii, makra, pliki nagłówkowe i pliki nagłówkowe), które definiują wartości stałych dostarczanych wraz z produktem IBM MQ for IBM i.
- Trzy programy pośredniczenia do łączenia-edycja z aplikacjami ILE C, ILE COBOL i ILE RPG.
- Pakiet przykładowych programów demonstrujący sposób korzystania z interfejsu MQI na platformie IBM i .

Produkty IBM MQ for Windows i IBM MQ w systemach UNIX and Linux również dostarczają:

- Wywołania, za pomocą których programy IBM MQ for Windows i IBM MQ w systemach UNIX and Linux mogą zatwierdzić i wycofać zmiany.
- *Uwzględnij pliki* , które definiują wartości stałych dostarczonych na tych platformach.
- *Pliki biblioteki* , aby połączyć aplikacje.
- Pakiet przykładowych programów, który demonstruje sposób użycia interfejsu MQI na tych platformach. Więcej informacji na temat tych przykładów można znaleźć w sekcji “Korzystanie z przykładowych programów na platformie Multiplatforms” na stronie 1166.
- Przykładowy kod źródłowy i kod wykonywalny dla powiązań z zewnętrznymi menedżerami transakcji.

Aby dowiedzieć się więcej na temat interfejsu MQI, należy użyć następujących odsyłaczy:

- “Wywołania MQI” na stronie 812
- “Wywołania punktu synchronizacji” na stronie 813
- “Konwersja danych, typy danych, definicje danych i struktury” na stronie 814
- “Programy kodu pośredniczącego i pliki bibliotek produktu IBM MQ” na stronie 814
- “Parametry wspólne dla wszystkich wywołań” na stronie 820
- “Określanie buforów” na stronie 821
- z/OS “Uwagi dotyczące zadań wsadowych z/OS” na stronie 821
- “Obsługa sygnału UNIX and Linux” na stronie 822

Pojęcia pokrewne

“Nawiąże połączenie z menedżerem kolejek i odłączenie go od niego” na stronie 824

Aby można było korzystać z usług programistycznych produktu IBM MQ , program musi mieć połączenie z menedżerem kolejek. Informacje zawarte w tej sekcji umożliwiają zapoznanie się z informacjami na temat nawiązywania połączenia z menedżerem kolejek i z jego rozłączeniem.

“Otwieranie i zamykanie obiektów” na stronie 833
Te informacje udostępniają wgląd w otwieranie i zamykanie obiektów produktu IBM MQ .

“Umieszczanie komunikatów w kolejce” na stronie 844
Te informacje umożliwiają zapoznanie się z informacjami na temat umieszczania komunikatów w kolejce.

“Pobieranie komunikatów z kolejki” na stronie 860
Ta sekcja zawiera informacje na temat pobierania komunikatów z kolejki.

“Sprawdzanie i ustawianie atrybutów obiektu” na stronie 943
Atrybuty to właściwości, które definiują parametry obiektu IBM MQ .

“Zatwierdzanie i wycofywanie jednostek pracy” na stronie 946
W tej sekcji opisano sposób zatwierdzania i tworzenia kopii zapasowych wszystkich możliwych do odtworzenia operacji get i put, które wystąpiły w jednostce pracy.

“Uruchamianie aplikacji produktu IBM MQ przy użyciu wyzwalaczy” na stronie 958
Informacje na temat wyzwalaczy i sposobów uruchamiania aplikacji IBM MQ przy użyciu wyzwalaczy.

“Praca z interfejsem MQI i klastrami” na stronie 978
Istnieją specjalne opcje dotyczące wywołań i kodów powrotu, które odnoszą się do technologii klastrowej.

“Używanie i zapisywanie aplikacji w systemie IBM MQ for z/OS” na stronie 983
Aplikacje produktu IBM MQ for z/OS mogą być uruchamiane z programów działających w wielu różnych środowiskach. Oznacza to, że mogą skorzystać z udogodnień dostępnych w więcej niż jednym środowisku.

“Aplikacje pomostowe IMS i IMS w systemie IBM MQ for z/OS” na stronie 70
Te informacje ułatwiają pisanie aplikacji produktu IMS przy użyciu produktu IBM MQ.

Wywołania MQI

Te informacje umożliwiają zapoznanie się z wywołaniami w interfejsie kolejki komunikatów (Message Queue Interface-MQI).

Wywołania w interfejsie MQI mogą być pogrupowane w następujący sposób:

MQCONN, MQCONNX i MQDISC
Za pomocą tych wywołań można połączyć program z (z opcjami lub bez) i odłączyć program od menedżera kolejek. Jeśli programy CICS są zapisywane w systemie z/OS, nie ma potrzeby korzystania z tych wywołań. Zaleca się jednak korzystanie z nich, jeśli aplikacja ma zostać portów aplikacji na innych platformach.

MQOPEN i MQCLOSE
Te wywołania umożliwiają otwarcie i zamknięcie obiektu, takiego jak kolejka.

MQPUT i MQPUT1
Użyj tych wywołań, aby umieścić komunikat w kolejce.

MQGET
Za pomocą tego wywołania można przeglądać komunikaty w kolejce lub usuwać komunikaty z kolejki.

MQSUB, MQSUBRQ
Te wywołania umożliwiają zarejestrowanie subskrypcji tematu i żądanie publikacji zgodnych z subskrypcją.

MQINQ
Użyj tego wywołania, aby dowiedzieć się o atrybutach obiektu.

MQSET
To wywołanie umożliwia ustawienie niektórych atrybutów kolejki. Nie można ustawić atrybutów innych typów obiektów.

MQBEGIN, MQCMIT i MQBACK
Należy użyć tych wywołań, gdy IBM MQ jest koordynatorem jednostki pracy. Komenda MQBEGIN uruchamia jednostkę pracy. MQCMIT i MQBACK kończą jednostkę pracy, zatwierdzając lub wycofując

aktualizacje wprowadzone podczas jednostki pracy. IBM i Kontroler transakcji IBM i jest używany do koordynowania globalnych jednostek pracy w systemie IBM MQ for IBM i. Używane są komendy rodzimego uruchamiania 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 na uchwyt komunikatu oraz usunąć uchwyt komunikatu.

MQSETMP, MQINQMP, MQDLTMP
Te wywołania umożliwiają ustawienie właściwości komunikatu na uchwycie komunikatu, zapytanie o właściwość komunikatu i usunięcie właściwości z uchwytu komunikatu.

MQCB, MQCB_FUNCTION, MQCTL
Te wywołania umożliwiają zarejestrowanie i sterowanie funkcją zwrotną.

MQSTAT
To wywołanie służy do pobierania informacji o statusie poprzednich asynchronicznych operacji put.

Opis wywołań MQI zawiera sekcja Opisy wywołań .

Wywołania punktu synchronizacji

Te informacje umożliwiają znalezienie informacji na temat wywołań punktu 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.

Za pomocą tych wywołań w programach wsadowych produktu z/OS należy poinformować menedżera kolejek o tym, że wszystkie operacje MQGET i MQPUT od ostatniego punktu synchronizacji mają zostać wykonane na stałe (zatwierdzone) lub mają być wycofane. Aby zatwierdzić zmiany i wycofać zmiany w innych środowiskach:

CICS
Użyj komend, takich jak EXEC CICS SYNCPOINT i EXEC CICS SYNCPOINT ROLLBACK.

IMS
Użyj obiektów punktu synchronizacji IMS , takich jak GU (pobierz unikalne) do wywołań IOPCB, CHKP (checkpoint) i ROLB (wycofywanie zmian).

RRS
Użyj odpowiednio: MQCMIT i MQBACK lub SRRCMIT i SRRBACK. (Patrz sekcja “Usługi zarządzania transakcjami i odtwarzalne usługi menedżera zasobów” na stronie 951).

Uwaga: Komendy SRRCMIT i SRRBACK są rodzimymi komendami RRS, ale nie są to wywołania MQI.

IBM i wywołania

IBM i

Produkt IBM MQ for IBM i udostępnia komendy MQCMIT i MQBACK. Można również użyć komend IBM i COMMIT i ROLLBACK lub dowolnych innych komend lub wywołań, które inicjują urządzenia kontroli transakcji IBM i (na przykład EXEC CICS SYNCPOINT).

Wywołania programu IBM MQ na platformach Windows, UNIX and Linux

ULW

Następujące produkty udostępniają wywołania MQCMIT i MQBACK:

- IBM MQ for Windows
- IBM MQ w systemach UNIX and Linux

Użyj wywołań punktu synchronizacji w programach, aby poinformować menedżera kolejek o tym, że wszystkie operacje MQGET i MQPUT od ostatniego punktu synchronizacji mają zostać wykonane na stałe (zatwierdzone) lub mają być wycofane. Aby zatwierdzić zmiany 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

Ta sekcja zawiera informacje na temat konwersji danych, podstawowych typów danych, definicji danych produktu IBM MQ i struktur w przypadku korzystania z interfejsu kolejki komunikatów.

Konwersja danych

Wywołanie MQXCNVC (przekształcanie znaków) przekształca dane znakowe komunikatu z jednego zestawu znaków na inny. Z wyjątkiem IBM MQ for z/OS, wywołanie to jest używane tylko z wyjścia konwersji danych.

Informacje na temat składni używanej w wywołaniu MQXCNVC zawiera sekcja MQXCNVC-Convert characters (MQXCNVC-Convert characters), a w celu uzyskania wskazówek dotyczących zapisywania i wywoływania wyjść konwersji danych, należy zapoznać się z “Pisanie wyjść konwersji danych” na stronie 1082 .

Elementarne typy danych

W przypadku obsługiwanych języków programowania interfejs MQI udostępnia elementarne typy danych lub pola nieustrukturyzowane.

Te typy danych są w pełni opisane w sekcji Typy danych elementarnych.

IBM MQ Definicje danych

z/OS Produkt IBM MQ for z/OS udostępnia definicje danych w postaci plików kopii w języku COBOL, makr w języku asemblacji, pojedynczego pliku włączanego języka PL/I, pojedynczego pliku włączanego w języku C oraz plików włączanych w języku C + +.

IBM i Produkt IBM MQ for IBM i udostępnia definicje danych w postaci plików kopii w języku COBOL, plików kopii RPG, plików włączanych w języku C oraz plików włączanych w języku C + +.

Pliki definicji danych dostarczane wraz z produktem IBM MQ zawierają następujące elementy:

- Definicje wszystkich stałych IBM MQ i kodów powrotu
- Definicje struktur i typów danych produktu IBM MQ
- Stałe definicje inicjowania struktur
- Prototypy funkcji dla każdego z wywołań (tylko dla języka PL/I i języka C)

Pełny opis plików definicji danych produktu IBM MQ można znaleźć w sekcji “Pliki definicji danych produktu IBM MQ” na stronie 806.

Struktury

Struktury, używane z wywołaniami MQI wymienionymi w programie “Wywołania MQI” na stronie 812, są dostarczane w plikach definicji danych dla każdego obsługiwanego języka programowania.
IBM i z/OS Pliki dostaw IBM MQ for z/OS i IBM MQ for IBM i zawierające stałe, które mają być używane podczas wypełniania niektórych pól tych struktur. Więcej informacji na ten temat zawiera sekcja Definicje danych produktu IBM MQ.

Podsumowanie struktur zawiera sekcja Podsumowanie typów danych struktury .

Programy kodu pośredniczącego i pliki bibliotek produktu IBM MQ

W tym miejscu wyświetlane są programy pośredniczenia i pliki bibliotek dla każdej platformy.

Więcej informacji na temat korzystania z programów pośredniczących i plików bibliotek podczas budowania aplikacji wykonywalnej zawiera sekcja “Budowanie aplikacji proceduralnej” na stronie 1099.

Informacje na temat tworzenia dowiązań do plików biblioteki C++ zawiera sekcja Korzystanie z języka C++ *IBM MQ Using C++*(Używanie języka C++ w języku C++).

AIX *IBM MQ for AIX pliki biblioteki*

W systemie IBM MQ for AIXnależy połączyć program z plikami biblioteki MQI dostarczonym dla środowiska, w którym aplikacja jest uruchamiana, oprócz tych udostępnionych przez system operacyjny.

W aplikacji, która nie jest wielowątkowa, należy utworzyć odsyłacz do jednej z następujących bibliotek:

Tabela 109. Pliki bibliotek dla niewielowątkowych aplikacji produktu AIX

Zbiór biblioteki	**Środowisko**
libmqm.a	Serwer dla C
libmqic.a i libmqm.a	Klient dla C
libmqmzf.a	Możliwe do zainstalowania wyjścia usługi dla języka C
libmqmxa.a	Interfejs XA serwera
libmqmxa64.a	Alternatywny interfejs XA serwera
libmqcxa.a	Interfejs klienta XA
libmqcxa64.a	Alternatywny interfejs XA klienta
libmqmcbrt.o	Biblioteka środowiska wykonawczego produktu IBM MQ dla obsługi Micro Focus COBOL
libmqmcb.a	Serwer dla języka COBOL
libmqicb.a	Klient dla języka COBOL
libimqc23ia.a	Klient dla C++
libimqs23ia.a	Serwer dla języka C++

W aplikacji wielowątkowej należy połączyć się z jedną z następujących bibliotek:

Tabela 110. Pliki bibliotek dla wielowątkowych aplikacji produktu AIX.

Tabela z dwiema kolumnami, w której znajdują się pliki bibliotek i środowisko dla każdego pliku biblioteki.

Zbiór biblioteki	**Środowisko**
libmqm_r.a	Serwer dla C
libmqic_r.a i libmqm_r.a	Klient dla C
libmqmzf_r.a	Możliwe do zainstalowania wyjścia usługi dla języka C
libmqmxa_r.a	Interfejs XA serwera
libmqmxa64_r.a	Alternatywny interfejs XA serwera
libmqcxa_r.a	Interfejs klienta XA
libmqcxa64_r.a	Alternatywny interfejs XA klienta
libimqc23ia_r.a	Klient dla C++
libimqs23ia_r.a	Serwer dla języka C++

Uwaga: Nie można połączyć się z więcej niż jedną biblioteką. Oznacza to, że nie można jednocześnie połączyć się z biblioteką wielowątkową i niewielowątkową.

IBM i *IBM MQ for IBM i pliki biblioteki*

W programie IBM MQ for IBM inależy połączyć program z plikami biblioteki MQI dostarczonym dla środowiska, w którym aplikacja jest uruchamiana, oprócz tych udostępnionych przez system operacyjny.

Dla aplikacji niewielowątkowych:

Tabela 111. Pliki bibliotek dla niewielowątkowych aplikacji produktu IBM i

Zbiór biblioteki	**Środowisko**
LIBMQM	Program usługowy serwera i klienta
LIBMQIC	Program obsługi klienta
IMQB23I4	Podstawowy program usługowy C++
IMQS23I4	Program usługowy serwera C++
LIBMQMZF	Wyjścia instalacyjne dla C

W aplikacji wielowątkowej:

Tabela 112. Pliki bibliotek dla wielowątkowych aplikacji produktu IBM i

Zbiór biblioteki	**Środowisko**
LIBMQM_R	Program usługowy serwera i klienta
IMQB23I4_R	Podstawowy program usługowy C++
IMQS23I4_R	Program usługowy serwera C++
LIBMQMZF_R	Wyjścia instalacyjne dla C
LIBMQIC_R	Program obsługi klienta

W programie IBM MQ for IBM imożna pisać aplikacje w języku C + +. Aby zobaczyć, jak połączyć aplikacje C++ i uzyskać szczegółowe informacje na temat wszystkich aspektów korzystania z języka C + +, należy zapoznać się z informacjami w sekcji Korzystanie z języka C++.

Linux *Pliki bibliotek produktu IBM MQ for Linux*

W systemie IBM MQ for Linuxnależy połączyć program z plikami biblioteki MQI dostarczonym dla środowiska, w którym aplikacja jest uruchamiana, oprócz tych udostępnionych przez system operacyjny.

W aplikacji, która nie jest wielowątkowa, należy utworzyć odsyłacz do jednej z następujących bibliotek:

Tabela 113. Pliki bibliotek dla niewielowątkowych aplikacji produktu Linux

Zbiór biblioteki	**Środowisko**
libmqm.so	Serwer dla C
libmqic.so i libmqm.so	Klient dla C
libmqmzf.so	Możliwe do zainstalowania wyjścia usługi dla języka C
libmqmxa.so	Interfejs XA serwera
libmqmxa64.so	Alternatywny interfejs XA serwera
libmqcxa.so	Interfejs klienta XA

Tabela 113. Pliki bibliotek dla niewielowątkowych aplikacji produktu Linux (kontynuacja)	
Zbiór biblioteki	**Środowisko**
libmqcxa64.so	Alternatywny interfejs XA klienta
libimqc23gl.so	Klient dla C++
libimqs23gl.so	Serwer dla języka C++

W aplikacji wielowątkowej należy połączyć się z jedną z następujących bibliotek:

Tabela 114. Pliki bibliotek dla wielowątkowych aplikacji produktu Linux	
Zbiór biblioteki	**Środowisko**
libmqm_r.so	Serwer dla C
libmqic_r.so i libmqm_r.so	Klient dla C
libmqmzf_r.so	Możliwe do zainstalowania wyjścia usługi dla języka C
libmqmxa_r.so	Interfejs XA serwera
libmqmxa64_r.so	Alternatywny interfejs XA serwera
libmqcxa_r.so	Interfejs klienta XA
libmqcxa64_r.so	Alternatywny interfejs XA klienta
libimqc23gl_r.so	Klient dla C++
libimqs23gl_r.so	Serwer dla języka C++

Uwaga: Nie można połączyć się z więcej niż jedną biblioteką. Oznacza to, że nie można jednocześnie połączyć się z biblioteką wielowątkową i niewielowątkową.

Solaris *IBM MQ for Solaris pliki biblioteki*

W systemie IBM MQ for Solarisnależy połączyć program z plikami biblioteki MQI dostarczonym dla środowiska, w którym działa aplikacja, oprócz tych, które są udostępniane przez system operacyjny.

Tabela 115. Pliki bibliotek dla aplikacji produktu Solaris	
Plik biblioteki	**Środowisko**
libmqm.so	Serwer i klient dla języka C
libmqmzse.so	Dla C
libmqic.so	Klient dla C
libmqmcs.so	Wspólne usługi dla C
libmqmzf.so	Możliwe do zainstalowania wyjścia usługi dla języka C
libmqmxa.so	Interfejs XA serwera
libmqmxa64.so	Alternatywny interfejs XA serwera
libmqcxa.so	Interfejs klienta XA
libmqcxa64.so	Alternatywny interfejs XA klienta
libimqc23as.a	Klient dla C++
libimqs23as.a	Serwer dla języka C++

Windows *IBM MQ for Windows pliki biblioteki*

W systemie IBM MQ for Windowsnależy połączyć program z plikami biblioteki MQI dostarczonym dla środowiska, w którym jest uruchamiana aplikacja, oprócz tych udostępnionych przez system operacyjny:

Tabela 116. Pliki bibliotek dla aplikacji produktu Windows

Plik biblioteki	**Środowisko**
MQ_INSTALLATION_PATH\Tools\Lib\mqm.lib	Serwer dla C (32-bitowa)
MQ_INSTALLATION_PATH\Tools\Lib\mqic.lib	Klient dla C (32-bitowa)
MQ_INSTALLATION_PATH\Tools\Lib\mqmxa.lib	Interfejs XA serwera dla języka C (32-bitowego)
MQ_INSTALLATION_PATH\Tools\Lib\mqcxa.lib	Interfejs klienta XA dla języka C (32-bitowego)
MQ_INSTALLATION_PATH\Tools\Lib\mqicxa.lib	Klient MTS dla C (32-bit)
MQ_INSTALLATION_PATH\Tools\Lib\mqmcics4.lib32	Obsługa serwera TXSeries CICS dla języka C (wersja 32-bitowa)
MQ_INSTALLATION_PATH\Tools\Lib\mqccics4.lib32	Obsługa klienta TXSeries CICS dla języka C (32-bitowego)
MQ_INSTALLATION_PATH\Tools\Lib\mqmzf.lib	Wyjścia usług instalowalnych dla języka C (32-bitowa)
MQ_INSTALLATION_PATH\Tools\Lib\mqmcbb.lib	Serwer dla IBM COBOL (wersja 32-bitowa)
MQ_INSTALLATION_PATH\Tools\Lib\mqmcb.lib	Serwer dla Micro Focus COBOL (32-bit)
MQ_INSTALLATION_PATH\Tools\Lib\mqiccbb.lib	Klient dla IBM COBOL (wersja 32-bitowa)
MQ_INSTALLATION_PATH\Tools\Lib\mqiccb.lib	Klient dla Micro Focus COBOL (32-bit)
MQ_INSTALLATION_PATH\Tools\Lib\imqs23vn.lib	Serwer dla języka C++ (32-bitowa)
MQ_INSTALLATION_PATH\Tools\Lib\imqc23vn.lib	Klient dla języka C++ (32-bitowa)
MQ_INSTALLATION_PATH\Tools\Lib\imqb23vn.lib	Podstawowa dla języka C++ (32-bitowa)
MQ_INSTALLATION_PATH\Tools\Lib\imqx23vn.lib	Klient MTS dla C++ (32-bitowa)
MQ_INSTALLATION_PATH\Tools\Lib64\mqm.lib	Serwer dla języka C (64-bitowy)
MQ_INSTALLATION_PATH\Tools\Lib64\mqic.lib	Klient dla języka 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	Interfejs klienta XA dla języka C (64-bitowy)
MQ_INSTALLATION_PATH\Tools\Lib64\mqicxa.lib	Klient MTS for C (64-bitowy)
MQ_INSTALLATION_PATH\Tools\Lib64\mqmcbb.lib	Serwer dla produktu 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	Klient dla produktu IBM COBOL (64-bitowy)
MQ_INSTALLATION_PATH\Tools\Lib64\mqiccb.lib	Klient dla Micro Focus COBOL (64-bitowy)
MQ_INSTALLATION_PATH\Tools\Lib64\imqs23vn.lib	Serwer dla C++ (64-bitowy)
MQ_INSTALLATION_PATH\Tools\Lib64\imqc23vn.lib	Klient dla C++ (64-bitowy)

Tabela 116. Pliki bibliotek dla aplikacji produktu Windows (kontynuacja)	
Plik biblioteki	**Środowisko**
MQ_INSTALLATION_PATH\Tools\Lib64\imqb23vn.lib	Podstawowy dla C++ (64-bitowy)
MQ_INSTALLATION_PATH\Tools\Lib64\imqx23vn.lib	Klient MTS dla C++ (64-bitowy)

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Program `amqmdnet.dll` służy do kompilowania programów .NET . Więcej informacji na ten temat zawiera sekcja “Kompilowanie programów IBM MQ .NET” na stronie 619 w sekcji “Tworzenie aplikacji produktu .NET” na stronie 566 .

Pliki te są dostarczane pod kątem zgodności z poprzednimi wersjami:

```
mqic32.lib
mqic32xa.lib
```

z/OS *Programy kodu pośredniczącego produktu IBM MQ for z/OS*

Zanim będzie możliwe uruchomienie programu napisanego z programem IBM MQ for z/OS, należy go połączyć z programem pośredniczonym dostarczonym z produktem IBM MQ for z/OS dla środowiska, w którym aplikacja jest uruchamiana.

Program pośredniczący udostępnia pierwszy etap przetwarzania wywołań w żądaniach, które mogą być przetwarzane przez program IBM MQ for z/OS .

Produkt IBM MQ for z/OS udostępnia następujące programy pośredniczenia:

SZKIEŁ_BD
Program pośredniczący dla programów wsadowych z/OS

CSQBRRSI
Program pośredniczący dla programów wsadowych produktu z/OS przy użyciu usługi RRS przy użyciu interfejsu MQI

CSQBRSTB
Program pośredniczący dla programów wsadowych z/OS za pomocą usługi RRS bezpośrednio

CSQCSTUB
Program pośredniczący dla programów CICS

KOD POŚREDNICZĄCY CSQQXX_ENCODE_CASE_CAPS_LOCK_OFF
Program pośredniczący dla programów IMS

KOD POŚREDNICZĄCY CSQX
Program pośredniczący dla rozproszonego kolejkowania wyjść innych niżCICS

CSQASTUB
Program pośredniczący dla wyjść konwersji danych

Ostrzeżenie: Jeśli używany jest program pośredniczący inny niż określony dla określonego środowiska, może on mieć nieprzewidywalne wyniki.

Uwaga: Jeśli używany jest program pośredniczący CSQBRSTB, link-edit z ATRSCSS z SYS1.CSSLIB. (SYS1.CSSLIB jest również znana jako *Biblioteka usług wywoływalnych*). Więcej informacji na temat usługi RRS zawiera sekcja “Usługi zarządzania transakcjami i odtwarzalne usługi menedżera zasobów” na stronie 951.

Alternatywnie można dynamicznie wywoływać kod pośredniczący z poziomu programu. Ta technika jest opisana w podręczniku “Dynamiczne wywoływanie kodu pośredniczącego produktu IBM MQ” na stronie 1136.

W programie IMSmoże być również konieczne użycie modułu interfejsu języka specjalnego, który jest dostarczany przez produkt IBM MQ.

Nie należy uruchamiać aplikacji, które są edytowane za pomocą odsyłaczy z kodem CSQBSTUB i CSQQSTUB w tym samym regionie MPP produktu 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 być uruchamiane w różnych regionach komunikatów produktu IMS .

Parametry wspólne dla wszystkich wywołań

Istnieją dwa typy parametrów wspólne dla wszystkich wywołań: uchwyty i kody powrotu.

Używanie uchwytów

Wszystkie wywołania MQI używają jednego lub więcej *uchwytów*. Identyfikują one menedżer kolejek, kolejkę lub inny obiekt, komunikat lub subskrypcję, odpowiednio do wywołania.

Aby program mógł komunikować się z menedżerem kolejek, program musi mieć unikalny identyfikator, za pomocą którego wie on, że menedżer kolejek jest poprawny. Ten identyfikator jest nazywany *uchwytem połączenia*, czasem nazywanych *Hconn*. W przypadku programów CICS uchwyt połączenia jest zawsze równy 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 korzystają z innych wywołań.

Aby program działał z obiektem IBM MQ , program musi mieć unikalny identyfikator, za pomocą którego obiekt ten wie, że obiekt jest obiektem. Ten identyfikator jest nazywany *uchwytem obiektu*, czasem nazywanych *Hobj*. Uchwyt jest zwracany przez wywołanie MQOPEN, gdy program otworzy obiekt do pracy z nim. Programy przekazują uchwyt obiektu jako parametr wejściowy, gdy korzystają z 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 przetwarzania komunikatów wymagają użycia *uchwytu komunikatu* lub *Hmsg*.

Kody powrotu zrozumienia

Kod zakończenia i kod przyczyny są zwracane przez każde wywołanie jako parametry wyjściowe. Są to znane zbiorczo *kody powrotu*.

Aby określić, czy wywołanie powiodło się, każde wywołanie zwraca *kod zakończenia* po zakończeniu wywołania. Kod zakończenia zwykle ma wartość MQCC_OK wskazującą powodzenie lub MQCC_FAILED wskazuje niepowodzenie. Niektóre wywołania mogą zwrócić stan pośredni, MQCC_WARNING, wskazując częściowy sukces.

Każde wywołanie zwraca także *kod przyczyny* , który przedstawia przyczynę niepowodzenia wywołania lub jego częściowy sukces. Istnieje wiele kodów przyczyny, obejmujących takie okoliczności, jak kolejka jest pełna, operacje pobierania nie są dozwolone dla kolejki, a konkretna kolejka nie jest zdefiniowana dla menedżera kolejek. Programy mogą używać kodu przyczyny w celu podjęcia decyzji o sposobie postępowania. Na przykład użytkownicy mogą podpowiadać użytkownikom zmianę danych wejściowych, a następnie ponownie wywołać wywołanie lub zwrócić użytkownikowi komunikat o błędzie.

Jeśli kod zakończenia ma wartość MQCC_OK, kodem przyczyny jest zawsze MQRC_NONE.

Kody zakończenia i przyczyny dla każdego wywołania są wymienione wraz z opisem tego wywołania. Zapoznaj się z opisami wywołań i wybierz odpowiednie wywołanie z listy.

Więcej szczegółowych informacji, w tym pomysłów na działania naprawcze, można znaleźć w:

- 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 pozostałych 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 wartość zerową, można użyć pustego wskaźnika do buforu.

Zawsze używaj długości fali podczas określania wielkości buforu, który jest wymagany.

Jeśli używany jest bufor do przechowywania danych wyjściowych z wywołania (na przykład w celu przechowywania danych komunikatu dla wywołania MQGET lub wartości atrybutów, których dotyczy wywołanie MQINQ), menedżer kolejek próbuje zwrócić kod przyczyny, jeśli podany bufor nie jest poprawny lub jest w pamięci masowej tylko do odczytu. Może jednak nie zawsze być w stanie zwrócić kod przyczyny.

z/OS *Uwagi dotyczące zadań wsadowych z/OS*

Programy wsadowe programu z/OS , które wywołują interfejs MQI, mogą znajdować się w stanie nadzorczym lub w stanie problemu.

Muszą one jednak spełniać następujące warunki:

- Muszą być one w trybie zadania, a nie w trybie blokady żądania obsługi (SRB).
- Muszą one znajdować się w trybie ASC (Primary Address Space Control), a nie w trybie dostępu do rejestru dostępu (Access Register ASC).
- Nie mogą one być w trybie pamięci krzyżowej. Podstawowy numer przestrzeni adresowej (ASN) musi być równy drugorzędnym ASN i domowym ASN.
- Nie mogą one być używane jako programy obsługi wyjścia MPF.
- Żadne blokady z/OS nie mogą być wstrzymane.
- Na stosie FRR nie może być żadnych procedur odtwarzania funkcji (FRR).
- Dla wywołania MQCONN lub MQCONNX mogą być używane dowolne słowo statusu programu (PSW) (pod warunkiem, że klucz jest kompatybilny z użyciem pamięci masowej, która znajduje się w kluczu TCB), ale kolejne wywołania używające uchwytu połączenia zwróconego przez produkt MQCONN lub MQCONNX:
  - Musi mieć ten sam klucz PSW, który był używany w wywołaniu MQCONN lub MQCONNX
  - Muszą mieć parametry dostępne (do zapisu, w stosownych przypadkach) pod tym samym kluczem 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 którekolwiek z tych warunków nie jest spełnione, może wystąpić sprawdzenie programu.
W niektórych przypadkach wywołanie nie powiedzie się i kod przyczyny zostanie zwrócony.

Linux UNIX *Uwagi dotyczące produktu UNIX and Linux*

Uwagi, o których należy pamiętać.

Podczas tworzenia aplikacji produktu UNIX and Linux należy pamiętać o następujących kwestiach.

Linux UNIX *Wywołanie systemowe fork w systemach UNIX and Linux*

Należy zwrócić uwagę na te uwagi w przypadku korzystania z wywołania systemowego fork w aplikacjach IBM MQ .

Jeśli aplikacja chce używać produktu `fork`, proces nadrzędny tej aplikacji powinien wywoływać produkt `fork` przed nawiązaniem wszystkich wywołań produktu IBM MQ , na przykład `MQCONN`, lub w celu utworzenia obiektu IBM MQ za pomocą programu **`ImqQueueManager`**.

Jeśli aplikacja chce utworzyć proces potomny po wprowadzeniu wszystkich wywołań produktu IBM MQ , kod aplikacji musi używać `fork()` z `exec()` , aby upewnić się, że jest to nowa instancja, a nie dokładna kopia elementu nadrzędnego.

Jeśli aplikacja nie korzysta z produktu `exec()`, wywołanie funkcji API IBM MQ wykonane w procesie potomnym zwraca wartość MQRC_ENVIRONMENT_ERROR.

Linux UNIX *Obsługa sygnału UNIX and Linux*

Nie dotyczy to IBM MQ for z/OS ani IBM MQ for Windows.

Ogólnie, systemy UNIX, Linux i IBM i zostały przeniesione z środowiska wielowątkowego do środowiska wielowątkowego. W środowisku bez gwintów niektóre funkcje mogą być zaimplementowane tylko za pomocą sygnałów, choć większość aplikacji nie musi być świadoma sygnałów i obsługi sygnałów. W środowisku wielowątkowym operacje podstawowe oparte na wątkach obsługują niektóre funkcje, które są używane do implementowania w środowiskach, które nie są gwintowane, przy użyciu sygnałów.

W wielu przypadkach sygnały i obsługa sygnałów, mimo że 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 (działaniowymi jako część aplikacji) w środowisku wielowątkowym, w którym każdy próbuje obsłużyć sygnały. Tradycyjne podejście do zapisywania i odtwarzania procedur obsługi sygnału (zdefiniowanych dla każdego procesu), które pracowało w sytuacji, gdy w procesie jest tylko jeden wątek wykonywania, nie działa w środowisku wielowątkowym. Jest to spowodowane tym, że wiele wątków wykonywania może próbować zapisać i odtworzyć zasób o zasięgu całego procesu, którego wyniki są nieprzewidywalne.

UNIX *Aplikacje wielowątkowe*

Nie ma zastosowania w systemie Solaris , ponieważ wszystkie aplikacje są traktowane jako wątki wielowątkowe, nawet jeśli używają tylko jednego wątku.

Każda funkcja MQI służy do konfigurowania własnej procedury obsługi sygnału dla sygnałów:

- SIGALRM
- SIGBUS
- SIGFPE
- SIGSEGV
- SIGILL

Procedury obsługi użytkowników dla tych elementów są zastępowane przez czas trwania wywołania funkcji MQI. Inne sygnały mogą być wychwytywać w normalny sposób przez osoby zajmujące się obsługą pisemną. Jeśli program obsługi nie zostanie zainstalowany, zostaną wykonane działania domyślne (na przykład ignorowanie, zrzut jądra lub wyjście).

Po IBM MQ obsługuje sygnał synchroniczny (SIGSEGV, SIGBUS, SIGFPE, SIGILL), który próbuje przekazać sygnał do dowolnej zarejestrowanej procedury obsługi sygnału, zanim wywoła wywołanie funkcji MQI.

Linux UNIX *Aplikacje gwintowane*

Wątek jest uznawany za połączony z produktem IBM MQ z tabeli MQCONN (lub MQCONNX) do momentu wywołania MQDISC.

Sygnały synchroniczne

Sygnały synchroniczne pojawiają się w określonym wątku.

Systemy UNIX and Linux umożliwiają bezpieczne ustawianie procedury obsługi sygnału dla takich sygnałów dla całego procesu. Program IBM MQ konfiguruje jednak własną procedurę obsługi dla następujących sygnałów w procesie aplikacji, podczas gdy dowolny wątek jest połączony z programem IBM MQ:

- SIGBUS
- SIGFPE
- SIGSEGV
- SIGILL

Jeśli piszesz aplikacje wielowątkowe, dla każdego sygnału istnieje tylko jedna procedura obsługi sygnału dla całego procesu. Gdy program IBM MQ konfiguruje własne procedury obsługi sygnału synchronicznego, zapisuje wszystkie wcześniej zarejestrowane procedury obsługi dla każdego sygnału. Po tym, jak program IBM MQ obsługuje jeden z wymienionych sygnałów, program IBM MQ próbuje wywołać procedurę obsługi sygnału, która była skuteczna w momencie pierwszego połączenia IBM MQ w procesie. Poprzednio zarejestrowane procedury obsługi są odtwarzane, gdy wszystkie wątki aplikacji zostały odłączone od produktu IBM MQ.

Ponieważ procedury obsługi sygnałów są składowane i odtwarzane przez program IBM MQ, wątki aplikacji nie mogą ustanawiać procedur obsługi sygnałów dla tych sygnałów, podczas gdy istnieje jakakolwiek 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) tworzy 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, generalna zasada jest taka, że ostatnia procedura obsługi sygnału, która ma być składowana, musi być pierwszą, która ma zostać odtworzona:

- Gdy aplikacja tworzy procedurę obsługi sygnału po nawiązaniu połączenia z produktem IBM MQ, poprzednia procedura obsługi sygnału musi zostać odtworzona, zanim aplikacja rozłączy się z programem IBM MQ.
- Gdy aplikacja tworzy procedurę obsługi sygnału przed nawiązaniem połączenia z serwerem IBM MQ, przed odtworzeniem procedury obsługi sygnału aplikacja musi odłączyć się od systemu IBM MQ .

Uwaga: Nieprzestrzeganie ogólnej zasady, zgodnie z którą ostatnia procedura obsługi sygnału, która ma być składowana, musi być pierwszą, która ma być odtworzona, może skutkować nieoczekiwaną obsługą sygnału w aplikacji i, potencjalnie, utratą sygnałów przez aplikację.

Sygnały asynchroniczne

Produkt IBM MQ nie używa żadnych sygnałów asynchronicznych w aplikacjach wielowątkowych, chyba że są to aplikacje klienckie.

Dodatkowe uwagi dotyczące wielowątkowych aplikacji klienckich

Produkt IBM MQ obsługuje następujące sygnały podczas operacji we/wy na serwerze. Sygnały te są definiowane przez stos komunikacji. Aplikacja nie może ustanawiać procedury obsługi sygnału dla tych sygnałów, gdy wątek jest połączony z menedżerem kolejek:

SIGPIPE (dla TCP/IP)

UNIX *Dodatkowe uwagi dotyczące korzystania z obsługi sygnałów UNIX w MQI*
Należy zapoznać się z tymi uwagami podczas korzystania z obsługi sygnałów UNIX .

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 “Nawiąże połączenie z menedżerem kolejek przy użyciu wywołania MQCONNX” na stronie 827.

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.

Nawiąże połączenie z menedżerem kolejek i odłączenie go od niego

Aby można było korzystać z usług programistycznych produktu IBM MQ , program musi mieć połączenie z menedżerem kolejek. Informacje zawarte w tej sekcji umożliwiają zapoznanie się z informacjami na temat nawiązywania połączenia z menedżerem kolejek i z jego rozłączeniem.

Sposób, w jaki to połączenie jest nawiązany, zależy od platformy i środowiska, w którym działa program:

Multi **IBM MQ for Multiplatforms**

Programy, które działają w tych środowiskach, mogą używać wywołania MQCONN MQI, z którym można nawiązać połączenie, oraz wywołania MQDISC, z którym ma zostać odłączone połączenie, a także menedżer kolejek. Alternatywnie programy mogą używać wywołania MQCONNX.

z/OS **IBM MQ for z/OS wsadowe**

Programy, które działają w tym środowisku, mogą używać wywołania MQCONN MQI, z którym ma zostać nawiązane połączenie, oraz wywołania MQDISC, z którym ma zostać odłączone połączenie, a także menedżer kolejek. Alternatywnie programy mogą używać wywołania MQCONNX.

Programy wsadowe z/OS mogą łączyć się, kolejno lub współbieżnie, z wieloma menedżerami kolejek w tym samym TCB.

z/OS **IMS**

Region sterujący IMS jest połączony z jednym lub większą liczbę menedżerów kolejek po jego uruchomieniu. To połączenie jest kontrolowane przez komendy IMS . Informacje na temat sposobu sterowania adapterem IMS w systemie z/OSzawiera sekcja Administrowanie programem IBM MQ for

z/OS. Jednak programy piszące kolejkowania komunikatów IMS muszą używać wywołania MQCONN MQI w celu określenia menedżera kolejek, z którym mają zostać nawiązane połączenie. Mogą one używać wywołania MQDISC do rozłączenia się z tym menedżerem kolejek.

Po wywołaniu programu IMS , który ustanawia punkt synchronizacji, oraz przed przetwornikiem komunikatu dla innego użytkownika, adapter IMS zapewnia, że aplikacja zamknie uchwyty i rozłącza się z menedżerem kolejek. Więcej informacji zawiera sekcja “Punkty synchronizacji w aplikacjach produktu IMS” na stronie 950.

Programy IMS mogą łączyć się, kolejno lub współbieżnie, z wieloma menedżerami kolejek w tym samym TCB.

z/OS CICS Transaction Server for z/OS

Programy CICS nie muszą wykonywać żadnej pracy w celu nawiązania połączenia z menedżerem kolejek, ponieważ sam system CICS jest połączony. To połączenie jest zwykle wykonywane automatycznie podczas inicjowania, ale można również użyć transakcji CKQC dostarczanej 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 połączony jest region CICS .

Programy CICS mogą również korzystać z wywołań MQI i rozłączania (MQCONN i MQDISC). Może to być konieczne, aby można było portów tych aplikacji w środowiskach innych niżCICS z minimum rekodowaniem. Jednak te wywołania *zawsze* pomyślnie zakończą się pomyślnie w środowisku CICS . Oznacza to, że kod powrotu może nie odzwierciedlać rzeczywistego stanu połączenia z menedżerem kolejek.

TXSeries dla systemów Windows i systemów otwartych

Programy te nie muszą wykonywać żadnej pracy w celu nawiązania połączenia z menedżerem kolejek, ponieważ jest on połączony z samym systemem CICS . W związku z tym obsługiwane jest tylko jedno połączenie w danym momencie. Aplikacje produktu CICS muszą wywoływać wywołanie MQCONN w celu uzyskania uchwytu połączenia, a wywołanie MQDISC przed ich wyjściem.

Użyj poniższych odsyłaczy, aby dowiedzieć się więcej na temat połączenia i rozłączenia z menedżerem kolejek:

- “Nawiąże połączenie z menedżerem kolejek przy użyciu wywołania MQCONN” na stronie 826
- “Nawiąże połączenie z menedżerem kolejek przy użyciu wywołania MQCONNX” na stronie 827
- “Rozłączanie programów z menedżera kolejek za pomocą MQDISC” na stronie 832

Pojęcia pokrewne

“Interfejs kolejki komunikatów-przegląd” na stronie 811
Informacje na temat komponentów interfejsu kolejek komunikatów (Message Queue Interface-MQI).

“Otwieranie i zamykanie obiektów” na stronie 833
Te informacje udostępniają wgląd w otwieranie i zamykanie obiektów produktu IBM MQ .

“Umieszczanie komunikatów w kolejce” na stronie 844
Te informacje umożliwiają zapoznanie się z informacjami na temat umieszczania komunikatów w kolejce.

“Pobieranie komunikatów z kolejki” na stronie 860
Ta sekcja zawiera informacje na temat pobierania komunikatów z kolejki.

“Sprawdzanie i ustawianie atrybutów obiektu” na stronie 943
Atrybuty to właściwości, które definiują parametry obiektu IBM MQ .

“Zatwierdzanie i wycofywanie jednostek pracy” na stronie 946
W tej sekcji opisano sposób zatwierdzania i tworzenia kopii zapasowych wszystkich możliwych do odtworzenia operacji get i put, które wystąpiły w jednostce pracy.

“Uruchamianie aplikacji produktu IBM MQ przy użyciu wyzwalaczy” na stronie 958
Informacje na temat wyzwalaczy i sposobów uruchamiania aplikacji IBM MQ przy użyciu wyzwalaczy.

“Praca z interfejsem MQI i klastrami” na stronie 978
Istnieją specjalne opcje dotyczące wywołań i kodów powrotu, które odnoszą się do technologii klastrowej.

“Używanie i zapisywanie aplikacji w systemie IBM MQ for z/OS” na stronie 983
Aplikacje produktu IBM MQ for z/OS mogą być uruchamiane z programów działających w wielu różnych środowiskach. Oznacza to, że mogą skorzystać z udogodnień dostępnych w więcej niż jednym środowisku.

“Aplikacje pomostowe IMS i IMS w systemie IBM MQ for z/OS” na stronie 70
Te informacje ułatwiają pisanie aplikacji produktu IMS przy użyciu produktu IBM MQ.

Nawiąże połączenie z menedżerem kolejek przy użyciu wywołania MQCONN

Informacje zawarte w tej sekcji umożliwiają poznanie sposobu łączenia się z menedżerem kolejek przy użyciu wywołania MQCONN.

W ogólnym przypadku można nawiązać połączenie z określonym menedżerem kolejek lub z domyślnym menedżerem kolejek:

- W przypadku systemu IBM MQ for z/OSw środowisku wsadowym domyślny menedżer kolejek jest określony w module CSQBDEFV.
- W systemach IBM MQ for Windows, IBM i, UNIXi Linux domyślny menedżer kolejek jest określony w pliku mqs.ini .

Alternatywnie w środowiskach z/OS MVS, TSO i RRS można nawiązać połączenie z dowolnym menedżerem kolejek w ramach grupy współużytkowania kolejek. Żądanie MQCONN lub MQCONNX wybiera jeden z aktywnych elementów grupy.

Po nawiązaniu połączenia z menedżerem kolejek musi on być lokalny względem zadania. Musi on należeć do tego samego systemu, co aplikacja IBM MQ .

W środowisku IMS menedżer kolejek musi być połączony z regionem sterującym IMS i z regionem zależnym, z którego korzysta program. Domyślny menedżer kolejek jest określany w module CSQQDEFV, gdy zainstalowany jest produkt IBM MQ for z/OS .

With the TXSeries CICS environment, and TXSeries for Windows and AIX, the queue manager must be defined as an XA resource to CICS.

Aby nawiązać połączenie z domyślnym menedżerem kolejek, należy wywołać MQCONN, określając nazwę składającą się całkowicie z odstępów lub rozpoczynając od znaku o wartości NULL (X'00 ').

Aplikacja musi być autoryzowana, aby można było pomyślnie nawiązać połączenie z menedżerem kolejek. Więcej informacji zawiera sekcja Bezpieczeństwo.

Dane wyjściowe MQCONN są następujące:

- Uchwyt połączenia (**Hconn**)
- Kod zakończenia
- Kod przyczyny

Użyj uchwytu połączenia przy kolejnych wywołaniach MQI.

Jeśli kod przyczyny wskazuje, że aplikacja jest już połączona z tym menedżerem kolejek, zwracany uchwyt połączenia jest taki sam, jak ten, który został zwrócony podczas pierwszego połączenia z aplikacją. Aplikacja nie może wywołać wywołania MQDISC w tej sytuacji, ponieważ aplikacja wywołująca oczekuje, że pozostanie połączona.

Zakres uchwytu połączenia jest taki sam, jak zasięg uchwytu obiektu (patrz “Otwieranie obiektów za pomocą wywołania MQOPEN” na stronie 834).

Opisy parametrów są podane w opisie wywołania MQCONN w MQCONN.

Wywołanie MQCONN kończy się niepowodzeniem, jeśli menedżer kolejek jest w stanie wygaszania podczas wydawania wywołania lub gdy menedżer kolejek jest zamykany.

Zasięg MQCONN lub MQCONNX

Zasięg wywołania MQCONN lub MQCONNX jest zwykle wątkiem, który go wydał. Oznacza to, że uchwyt połączenia zwrócony z wywołania jest poprawny tylko w obrębie wątku, który wywołał wywołanie. Tylko jedno połączenie może być wykonane w dowolnym momencie za pomocą uchwytu. Jeśli jest używany

z innego wątku, jest on odrzucany jako niepoprawny. Jeśli w aplikacji znajduje się wiele wątków, a każda z nich chce korzystać z wywołań programu IBM MQ , każda z nich musi wydać komendę MQCONN lub MQCONNX.

Nie jest konieczne, aby każde wywołanie było wykonywane w tym samym menedżerze kolejek, gdy proces wykonuje wiele wywołań MQCONN. Jednak tylko jedno połączenie IBM MQ może być wykonane z wątku w danym momencie. Alternatywnie można rozważyć użycie produktu "Połączenia współużytkowane (niezależne od wątku) z produktem MQCONNX" na stronie 830 w celu zezwolenia na użycie wielu połączeń IBM MQ z jednego wątku i połączenia IBM MQ z dowolnego wątku.[7]

Jeśli aplikacja działa jako klient, może on łączyć się z więcej niż jednym menedżerem kolejek w obrębie wątku.

Nawiąże połączenie z menedżerem kolejek przy użyciu wywołania MQCONNX

Wywołanie MQCONNX jest podobne do wywołania MQCONN, ale zawiera opcje służące do sterowania sposobem działania wywołania.

Jako dane wejściowe dla produktu MQCONNX można podać nazwę menedżera kolejek z/OS lub nazwę grupy współużytkowania kolejki w systemach współużytkowanych kolejek produktu z/OS.

Dane wyjściowe komendy MQCONNX są następujące:

- Uchwyt połączenia (Hconn)
- Kod zakończenia
- Kod przyczyny

Uchwyt połączenia jest używany podczas kolejnych wywołań MQI.

Opis wszystkich parametrów komendy MQCONNX jest podany w tabeli MQCONNX. Pole *Options* umożliwia ustawienie wartości STANDARD_BINDING, FASTPATH_BINDING, SHARED_BINDING lub ISOLATED_BINDING dla dowolnej wersji MQCNO. Za pomocą wywołania MQCONNX można również udostępnić współużytkowane (niezależne od wątku) połączenia. Więcej informacji na ten temat zawiera sekcja "Połączenia współużytkowane (niezależne od wątku) z produktem MQCONNX" na stronie 830 .

MQCNO_STANDARD_BINDING

Domyślnie MQCONNX (np. MQCONN) oznacza dwa wątki logiczne, w których aplikacja IBM MQ i agent lokalnego menedżera kolejek działają w oddzielnych procesach. Aplikacja IBM MQ żąda operacji IBM MQ , a lokalny agent menedżera kolejek żąda żądania. Wartość ta jest definiowana przez opcję MQCNO_STANDARD_BINDING w wywołaniu MQCONNX.

Jeśli zostanie określona wartość MQCNO_STANDARD_BINDING, wywołanie MQCONNX będzie używało wartości MQCNO_SHARED_BINDING lub MQCNO_ISOLATED_BINDING, w zależności od wartości atrybutu **DefaultBindType** menedżera kolejek zdefiniowanego w pliku qm.ini.

Jest to wartość domyślna.

W przypadku łączenia z biblioteką mqm najpierw podejmowana jest próba połączenia standardowego z serwerem przy użyciu domyślnego typu powiązania. Jeśli nie powiodła się próba załadowania bazowej biblioteki serwera, zostanie podjęta próba nawiązania połączenia z klientem.

- Jeśli określono zmienną środowiskową MQ_CONNECT_TYPE, można podać jedną z następujących opcji, aby zmienić zachowanie wartości MQCONN lub MQCONNX, jeśli określono wartość MQCNO_STANDARD_BINDING. (Wyjątkiem jest to, że parametr MQCNO_FASTPATH_BINDING został określony z parametrem MQ_CONNECT_TYPE ustawionym na wartość LOCAL lub STANDARD , aby zezwolić na obniżenie wartości połączeń fastpath przez administratora bez powiązanej zmiany z aplikacją:

[7] Jeśli używane są aplikacje wielowątkowe z produktem IBM MQ w systemach UNIX and 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 wykonują wywołania MQI samodzielnie lub z innymi procedurami obsługi sygnału (na przykład CICS).

Wartość	Znaczenie
KLIENT	Próba nawiązania połączenia z klientem jest wykonywana tylko przez klienta.
Krótka ścieżka	Ta wartość była obsługiwana w poprzednich wersjach, ale została zignorowana, jeśli została określona.
LOCAL	Próba nawiązania połączenia z serwerem jest wykonywana tylko przez serwer. Połączenia krótkiej ścieżki są obniżane do standardowego połączenia z serwerem.
STANDARDOWA	Obsługiwane w celu zapewnienia zgodności z poprzednimi wersjami. Ta wartość jest teraz traktowana jako LOCAL.

- Jeśli zmienna środowiskowa MQ_CONNECT_TYPE nie jest ustawiona, gdy wywoływana jest wartość MQCONN, podejmowana jest próba połączenia standardowego z serwerem przy użyciu domyślnego typu powiązania. Jeśli ładowanie biblioteki serwera nie powiedzie się, zostanie podjęta próba nawiązania połączenia z klientem.

MQCNO_FASTPATH_BINDING

Zaufane aplikacje oznacza, że aplikacja IBM MQ i agent lokalnego menedżera kolejek stają się tym samym procesem. Ponieważ proces agenta nie musi już korzystać z interfejsu w celu uzyskania dostępu do menedżera kolejek, te aplikacje stają się rozszerzeniem menedżera kolejek. Wartość ta jest definiowana przez opcję MQCNO_FASTPATH_BINDING w wywołaniu MQCONNX.

Należy połączyć zaufane aplikacje z wątkami bibliotek produktu IBM MQ . Instrukcje na temat sposobu konfigurowania aplikacji IBM MQ do uruchamiania jako zaufane zawiera sekcja Opcje MQCNO.

Ta opcja zapewnia najwyższą wydajność.

Uwaga: Ta opcja zapewnia zachowanie integralności menedżera kolejek: nie ma żadnej ochrony przed nadpisaniem jej pamięci masowej. Dotyczy to również sytuacji, gdy aplikacja zawiera błędy, które mogą być ujawnione dla komunikatów i innych danych w menedżerze kolejek. Przed skorzystaniem z tej opcji należy rozważyć następujące kwestie.

MQCNO_SHARED_BINDING

Tę opcję należy określić, aby aplikacja i agent lokalnego menedżera kolejek były uruchamiane w oddzielnych procesach. Zapewnia to zachowanie integralności menedżera kolejek, czyli chroni menedżer kolejek przed programami błędnymi. Jednak aplikacja i agent lokalnego menedżera kolejek współużytkują niektóre zasoby.

Ta opcja jest pośrednia między powiązaniem MQCNO_FASTPATH_BINDING i MQCNO_ISOLATED_BINDING, zarówno jeśli chodzi o ochronę integralności menedżera kolejek, jak i w zakresie wydajności wywołań MQI.

Wartość MQCNO_SHARED_BINDING jest ignorowana, jeśli menedżer kolejek nie obsługuje tego typu powiązania. Przetwarzanie jest kontynuowane tak, jakby opcja nie została określona.

Jeśli aplikacja nawiązała połączenie z lokalnym menedżerem kolejek za pomocą komendy MQCNO_SHARED_BINDING, menedżer kolejek może zostać zatrzymany, gdy aplikacja jest uruchomiona. Jeśli menedżer kolejek zostanie zrestartowany w czasie, gdy aplikacja nadal działa, próba uruchomienia menedżera kolejek zakończy się niepowodzeniem z błędem AMQ7018 , ponieważ aplikacja nadal wstrzymuje się do zasobów wymaganych przez menedżer kolejek.

Aby uruchomić menedżer kolejek, należy zatrzymać aplikację.

MQCNO_ISOLATED_BINDING

Tę opcję należy określić, aby aplikacja i agent lokalnego menedżera kolejek były uruchamiane w oddzielnych procesach, co w przypadku parametru MQCNO_SHARED_BINDING. Jednak w tym przypadku proces aplikacji i agent lokalnego menedżera kolejek są odizolowane od siebie, ponieważ nie współużytkują zasobów.

Jest to najbezpieczniejsza opcja zabezpieczania integralności menedżera kolejek, ale zapewnia najmniejszą wydajność wywołań MQI.

Wartość MQCNO_ISOLATED_BINDING jest ignorowana, jeśli menedżer kolejek nie obsługuje tego typu powiązania. Przetwarzanie jest kontynuowane tak, jakby opcja nie została określona.

MQCNO_CLIENT_BINDING

Tę opcję należy określić, aby aplikacja próbowała wykonać próbę nawiązania połączenia z klientem. Ta opcja ma następujące ograniczenia:

- z/OS Powiązanie MQCNO_CLIENT_BINDING jest ignorowane w systemie z/OS.
- Wartość MQCNO_CLIENT_BINDING jest odrzucana za pomocą wywołania MQRC_OPTIONS_ERROR, jeśli jest ona określona z dowolną opcją powiązania MQCNO inną niż MQCNO_STANDARD_BINDING.
- Powiązanie MQCNO_CLIENT_BINDING nie jest dostępne dla produktu Java , ponieważ ma własne mechanizmy wyboru typu powiązania.
- Jeśli zmienna środowiskowa MQ_CONNECT_TYPE nie jest ustawiona, gdy wywoływana jest wartość MQCONN, podejmowana jest próba połączenia standardowego z serwerem przy użyciu domyślnego typu powiązania. Jeśli ładowanie biblioteki serwera nie powiedzie się, zostanie podjęta próba nawiązania połączenia z klientem.

MQCNO_LOCAL_BINDING

Tę opcję należy określić, aby aplikacja próbowała nawiązać połączenie z serwerem. Jeśli określono również parametr MQCNO_FASTPATH_BINDING, MQCNO_ISOLATED_BINDING lub MQCNO_SHARED_BINDING, to połączenie jest typu tego typu i jest udokumentowane w tej sekcji. W przeciwnym razie zostanie podjęta próba użycia standardowego połączenia z serwerem przy użyciu domyślnego typu powiązania. Parametr MQCNO_LOCAL_BINDING ma następujące ograniczenia:

- z/OS Wartość MQCNO_LOCAL_BINDING jest ignorowana w systemie z/OS.
- Wartość MQCNO_LOCAL_BINDING jest odrzucana za pomocą wywołania MQRC_OPTIONS_ERROR, jeśli jest ona określona z dowolną opcją ponownego połączenia MQCNO, inną niż MQCNO_RECONNECT_AS_DEF.
- Opcja MQCNO_LOCAL_BINDING nie jest dostępna dla produktu Java , ponieważ ma własne mechanizmy wyboru typu powiązania.
- Jeśli zmienna środowiskowa MQ_CONNECT_TYPE nie jest ustawiona, gdy wywoływana jest wartość MQCONN, podejmowana jest próba połączenia standardowego z serwerem przy użyciu domyślnego typu powiązania. Jeśli ładowanie biblioteki serwera nie powiedzie się, zostanie podjęta próba nawiązania połączenia z klientem.

z/OS W systemie z/OS te opcje są tolerowane, ale wykonywane jest tylko połączenie standardowe.

z/OS Produkt MQCNO w wersji 3 dla produktu z/OS umożliwia cztery różne opcje:

MQCNO_SERIALIZE_CONN_TAG_QSG

Umożliwia to aplikacji żądanie, aby w grupie współużytkowania kolejki tylko jedna instancja aplikacji została uruchomiona w dowolnym momencie. Jest to osiągane przez zarejestrowanie użycia znacznika połączenia z wartością, która została określona lub wyprowadzona przez aplikację. Znacznik to 128-bajtowy łańcuch znaków określony w MQCNO w wersji 3.

MQCNO_RESTRICT_CONN_TAG_QSG

Ta opcja jest używana w przypadku, gdy aplikacja składa się z więcej niż jednego procesu (lub TCB), z których każdy może połączyć się z menedżerem kolejek. Połączenie jest dozwolone tylko wtedy, gdy nie ma bieżącego użycia znacznika lub aplikacja żądająca znajduje się w tym samym zasięgu przetwarzania. Jest to przestrzeń adresowa MVS w obrębie tej samej grupy współużytkowania kolejki, co właściciel znacznika.

MQCNO_SERIALIZE_CONN_TAG_Q_MGR

Jest to podobne do wywołania MQCNO_SERIALIZE_CONN_TAG_QSG, ale tylko lokalny menedżer kolejek jest interrogowany, aby sprawdzić, czy żądany znacznik jest już używany.

MQCNO_RESTRICT_CONN_TAG_Q_MGR

Jest to podobne do wartości MQCNO_RESTRICT_CONN_TAG_QSG, ale tylko lokalny menedżer kolejek jest interrogowany, aby sprawdzić, czy żądany znacznik jest już używany.

Ograniczenia dotyczące zaufanych aplikacji

Do zaufanych aplikacji mają zastosowanie następujące ograniczenia:

- Należy jawnie odłączyć zaufane aplikacje od menedżera kolejek.
- Przed zakończeniem działania menedżera kolejek za pomocą komendy endmqm należy zatrzymać zaufane aplikacje.
- Nie wolno używać sygnałów asynchronicznych i przerwań zegara (takich jak `sigkill`) z opcją MQCNO_FASTPATH_BINDING.
- Na wszystkich platformach wątek w zaufanej aplikacji nie może połączyć się z menedżerem kolejek, gdy inny wątek w tym samym procesie jest połączony z innym menedżerem kolejek.
- Linux UNIX W systemach UNIX and Linux należy używać mqm jako efektywnego userID i groupID dla wszystkich wywołań MQI. Te identyfikatory można zmienić przed wywołaniem uwierzytelniania innego niż MQI wymagającym uwierzytelniania (na przykład otwierając plik), ale przed dokonaniem następnego wywołania MQI należy zmienić tę wartość z powrotem na mqm.
- IBM i W systemie IBM i:
  1. Zaufane aplikacje muszą być uruchamiane w profilu użytkownika QMQM. Nie jest wystarczające, aby profil użytkownika był członkiem grupy QMQM lub aby program adoptowali uprawnienia QMQM. Możliwe, że profil użytkownika QMQM nie może być używany do wpisywania się do zadań interaktywnych lub do określenia w opisie zadania dla zadań uruchamiających zaufane aplikacje. W tym przypadku jednym podejściem jest użycie funkcji API wymiany profilu produktu IBM i , QSYGETPH, QWTSETP i QSYRLSPH, aby tymczasowo zmienić bieżącego użytkownika zadania na QMQM, podczas gdy programy MQ są uruchamiane. Szczegółowe informacje na temat tych funkcji wraz z przykładem ich użycia znajdują się w sekcji Funkcje API dotyczące zabezpieczeń w podręczniku *IBM i System API Reference*.
  2. Nie należy anulować zaufanych aplikacji za pomocą opcji 2 (System-Request) lub poprzez zakończenie zadań, w których są one uruchamiane za pomocą komendy ENDJOB.
- Windows Linux UNIX W systemach UNIX, Linux, and Windows zaufane 32-bitowe aplikacje nie są obsługiwane. W przypadku próby uruchomienia zaufanej 32-bitowej aplikacji zostanie ona obniżona do standardowego połączenia powiązanego.

Połączenia współużytkowane (niezależne od wątku) z produktem MQCONNX
Te informacje umożliwiają zapoznanie się ze współużytkowanymi połączeniami z tabelą MQCONNX i uwagami dotyczącymi użycia do rozważenia.

Uwaga: Nieobsługiwane w produkcie IBM MQ for z/OS.

Na platformach IBM MQ innych niż IBM MQ for z/OSpołączenie nawiązane z produktem `MQCONN` jest dostępne tylko dla wątku, który nawiąże połączenie. Opcje w wywołaniu MQCONNX pozwalają na utworzenie połączenia, które może być współużytkowane przez wszystkie wątki w procesie. Jeśli aplikacja

jest uruchomiona 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 korzystać z jednego z następujących niezależnych, współużytkowanych opcji połączeń:

MQCNO_HANDLE_SHARE_BLOCK
Tworzy połączenie współużytkowane. W przypadku połączenia z serwerem MQCNO_HANDLE_SHARE_BLOCK , jeśli połączenie jest obecnie używane przez wywołanie MQI w innym wątku, wywołanie MQI oczekuje do czasu zakończenia bieżącego wywołania MQI.

MQCNO_HANDLE_SHARE_NO_BLOCK
Tworzy połączenie współużytkowane. W przypadku połączenia z serwerem MQCNO_HANDLE_SHARE_NO_BLOCK , jeśli połączenie jest obecnie używane przez wywołanie MQI w innym wątku, wywołanie MQI nie powiedzie się natychmiast 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.

Z wywołania MQCONNX zwracany jest uchwyt połączenia. Uchwyt może być używany przez kolejne wywołania MQI z dowolnego wątku w procesie, tworząc powiązanie tych wywołań z uchwytem zwróconego z serwera MQCONNX. Wywołania MQI używające pojedynczego współużytkowanego uchwytu są serializowane między wątkami.

Na przykład następująca sekwencja działań jest możliwa przy użyciu uchwytu współużytkowanego:

1. Wątek 1 wydaje MQCONNX i pobiera 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 użyciu *h1*

Mimo że uchwyt jest używany przez dowolny wątek, dostęp do połączenia jest niedostępny dla innych wątków. W sytuacji, gdy dopuszczalne jest, że wątek oczekuje na zakończenie dowolnego poprzedniego wywołania z innego wątku, należy użyć opcji MQCONNX z opcją MQCNO_HANDLE_SHARE_BLOCK.

Jednak blokowanie może powodować trudności. Przypuśćmy, że w kroku “2” na stronie 831wątek 1 wysyła żądanie pobrania, które oczekuje na komunikaty, które mogły nie zostać jeszcze odebrane (a get with wait). W tym przypadku wątki 2 i 3 również pozostawiane są na czas oczekiwania (zablokowane) tak długo, jak długo trwa żądanie pobrania wątku 1. Jeśli wolisz, aby wywołanie MQI zostało zwrócone z błędem, jeśli na tym uchwycie jest już uruchomione inne wywołanie MQI, użyj opcji MQCONNX z opcją MQCNO_HANDLE_SHARE_NO_BLOCK.

Uwagi dotyczące użycia współużytkowanego połączenia

1. Wszystkie uchwyty obiektów (Hobj) utworzone przez otwarcie obiektu są powiązane z Hconn; tak dla wspólnego Hconn, Hobjs są również współużytkowane i możliwe do wykorzystania przez dowolny wątek z użyciem Hconn. Podobnie każda jednostka pracy rozpoczęta pod Hconn jest powiązana z tym Hconn, więc ta również jest współużytkowana przez wątki ze współużytkaną Hconn.
2. *Dowolny* wątek może wywołać komendę MQDISC, aby odłączyć współużytkowane Hconn, a nie tylko wątek, który wywołał odpowiednią tabelę MQCONNX. Zmaterializowana tabela MQDISC kończy działanie programu Hconn, które nie jest dostępne dla wszystkich wątków.
3. Pojedynczy wątek może używać wielu współużytkowanych zasobów Hconns szeregowo, na przykład za pomocą komendy MQPUT można umieścić jeden komunikat pod jednym współużytkowanym Hconn, a następnie umieścić kolejny komunikat przy użyciu innego współużytkowanego Hconn, przy czym każda operacja jest wykonywana w innej lokalnej jednostce pracy.

4. Współużytkowane połączenia Hconns nie mogą być używane w ramach globalnej jednostki pracy.

Użycie opcji wywołania MQCONNX z wartością MQ_CONNECT_TYPE
Te informacje umożliwiają zrozumienie różnych opcji wywołania MQCONNX i sposobu ich używania ze zmienną środowiskową MQ_CONNECT_TYPE.

Uwaga: Wartość MQ_CONNECT_TYPE ma wpływ tylko na powiązania STANDARD. W przypadku innych powiązań wartość MQ_CONNECT_TYPE jest ignorowana.

W systemach IBM MQ for IBM i, IBM MQ for Windowsi IBM MQ w systemach UNIX and Linux można użyć zmiennej środowiskowej MQ_CONNECT_TYPE w połączeniu z typem powiązania określonym w polu *Options* struktury MQCNO używanej w wywołaniu MQCONNX.

*Tabela 117. 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
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.

Rozłączanie programów z menedżera kolejek za pomocą MQDISC

Ta sekcja zawiera informacje na temat odłączania programów z menedżera kolejek za pomocą komendy MQDISC.

Gdy program, który nawiąże połączenie z menedżerem kolejek przy użyciu wywołania MQCONN lub MQCONNX, zakończy wszystkie interakcje z menedżerem kolejek, przerywa on połączenie za pomocą wywołania MQDISC, z wyjątkiem:

- W przypadku aplikacji CICS Transaction Server for z/OS , gdzie wywołanie jest opcjonalne, chyba że użyto komendy MQCONNX, a użytkownik chce usunąć znacznik połączenia przed zakończeniem aplikacji.
- W systemie IBM MQ for IBM i , w którym po wylogowaniu się z systemu operacyjnego zostanie wykonane niejawne wywołanie MQDISC.

Jako dane wejściowe wywołania MQDISC należy podać uchwyt połączenia (Hconn), który został zwrócony przez komendę MQCONN lub MQCONNX po nawiązaniu połączenia z menedżerem kolejek.

Z wyjątkiem CICS w systemie 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 funkcja MQCONN lub MQCONNX. Komenda MQDISC wykonuje niejawną operację MQCLOSE dla wszystkich obiektów, które nadal są otwarte za pomocą tego uchwytu.

z/OS W przypadku klienta połączonego z produktem z/OSpo wywołaniu wywołania MQDISC następuje niejawne zatwierdzenie, ale wszystkie uchwyty kolejki, które są nadal otwarte, nie są zamykane, dopóki kanał nie zostanie rzeczywiście zakończony.

Jeśli program MQCONNX jest używany do łączenia się z produktem IBM MQ for z/OS, to produkt MQDISC kończy również zasięg znacznika połączenia utworzonego przez produkt MQCONNX. Jednak w przypadku aplikacji CICS, IMSlub RRS, jeśli istnieje aktywna jednostka odtwarzania powiązana ze znacznikiem połączenia, MQDISC zostanie odrzucone z kodem przyczyny MQRC_CONN_TAG_NOT_RELEASED.

Opisy parametrów są podane w opisie wywołania MQDISC w MQDISC.

Jeśli nie zostanie wydana żadna wartość MQDISC

Standardowe, niewspółużytkowane połączenie (Hconn) jest czyszczone podczas końców tworzenia wątku. Połączenie współużytkowane jest tylko niejawnie wycofane i rozłączone, gdy cały proces kończy działanie. Jeśli wątek, który utworzył współużytkowany Hconn, kończy się, a Hconn nadal istnieje, to Hconn jest nadal użyteczny.

Sprawdzanie uprawnień

Wywołania MQCLOSE i MQDISC zwykle nie sprawdzają uprawnień.

W normalnym toku zdarzeń zadanie, które ma uprawnienie do otwierania lub nawiązywania połączenia z obiektem IBM MQ , zamyka się lub odłączy od tego obiektu. Nawet jeśli uprawnienie zadania, które nawiąże połączenie z obiektem IBM MQ lub go otworzyło, zostało odwołane, wywołania MQCLOSE i MQDISC są akceptowane.

Otwieranie i zamykanie obiektów

Te informacje udostępniają wgląd w otwieranie i zamykanie obiektów produktu IBM MQ .

Aby wykonać którekolwiek z poniższych operacji, należy najpierw *otworzyć* odpowiedni obiekt IBM MQ :

- Umieszczanie komunikatów w kolejce
- Pobieranie (przeglądanie lub pobieranie) komunikatów z kolejki
- Ustawianie atrybutów obiektu
- Zapytaj o atrybuty dowolnego obiektu

Użyj wywołania MQOPEN, aby otworzyć obiekt, korzystając z opcji wywołania w celu określenia, co ma być zrobione 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 “Umieszczanie jednego komunikatu w kolejce przy użyciu wywołania MQPUT1” na stronie 853).

Przed otwarniem obiektu za pomocą wywołania MQOPEN należy połączyć program z menedżerem kolejek. Jest to szczegółowo wyjaśnione w przypadku wszystkich środowisk w produkcie “Nawiąże połączenie z menedżerem kolejek i odłączenie go od niego” na stronie 824.

Istnieją cztery typy obiektów produktu IBM MQ , które można otworzyć:

- Kolejka
- Lista nazw
- Definicja procesu
- Menedżer kolejek

Wszystkie te obiekty można otworzyć w podobny sposób, korzystając z wywołania MQOPEN. Więcej informacji na temat obiektów IBM MQ zawiera sekcja Typy obiektów.

Ten sam obiekt można otworzyć więcej niż raz, a za każdym razem można uzyskać nowy uchwyt obiektu. Użytkownik może chcieć przeglądać komunikaty w kolejce za pomocą jednego uchwytu i usunąć komunikaty z tej samej kolejki za pomocą innego uchwytu. Umożliwia to składowanie przy użyciu zasobów do zamknięcia i ponownego otwarcia tego samego obiektu. Istnieje również możliwość otwarcia kolejki w celu przeglądania *i* usuwania komunikatów w tym samym czasie.

Ponadto istnieje możliwość otwarcia wielu obiektów za pomocą jednej operacji MQOPEN i zamknięcia ich za pomocą komendy MQCLOSE. Informacje o tym, jak to zrobić, zawiera sekcja “Lista dystrybucyjna” na stronie 854 .

Podczas próby otwarcia obiektu menedżer kolejek sprawdza, czy użytkownik jest uprawniony do otwarcia tego obiektu dla opcji określonych w wywołaniu MQOPEN.

Obiekty są zamykane automatycznie, gdy program rozłącza się z menedżerem kolejek. W środowisku IMS odłączenie jest wymuszane, gdy program rozpocznie przetwarzanie dla nowego użytkownika po

wywołaniu komendy IMS (get unique). Na platformie IBM i obiekty są zamykane automatycznie po zakończeniu zadania.

Dobrą praktyką programowania jest zamykanie otwartych obiektów. Aby to zrobić, należy użyć wywołania MQCLOSE.

Aby dowiedzieć się więcej na temat otwierania i zamykania obiektów, należy użyć następujących odsyłaczy:

- “Otwieranie obiektów za pomocą wywołania MQOPEN” na stronie 834
- “Tworzenie kolejek dynamicznych” na stronie 842
- “Otwieranie kolejek zdalnych” na stronie 843
- “Zamykanie obiektów przy użyciu wywołania MQCLOSE” na stronie 843

Pojęcia pokrewne

“Interfejs kolejki komunikatów-przegląd” na stronie 811
Informacje na temat komponentów interfejsu kolejek komunikatów (Message Queue Interface-MQI).

“Nawiąże połączenie z menedżerem kolejek i odłączenie go od niego” na stronie 824
Aby można było korzystać z usług programistycznych produktu IBM MQ , program musi mieć połączenie z menedżerem kolejek. Informacje zawarte w tej sekcji umożliwiają zapoznanie się z informacjami na temat nawiązywania połączenia z menedżerem kolejek i z jego rozłączeniem.

“Umieszczanie komunikatów w kolejce” na stronie 844
Te informacje umożliwiają zapoznanie się z informacjami na temat umieszczania komunikatów w kolejce.

“Pobieranie komunikatów z kolejki” na stronie 860
Ta sekcja zawiera informacje na temat pobierania komunikatów z kolejki.

“Sprawdzanie i ustawianie atrybutów obiektu” na stronie 943
Atrybuty to właściwości, które definiują parametry obiektu IBM MQ .

“Zatwierdzanie i wycofywanie jednostek pracy” na stronie 946
W tej sekcji opisano sposób zatwierdzania i tworzenia kopii zapasowych wszystkich możliwych do odtworzenia operacji get i put, które wystąpiły w jednostce pracy.

“Uruchamianie aplikacji produktu IBM MQ przy użyciu wyzwalaczy” na stronie 958
Informacje na temat wyzwalaczy i sposobów uruchamiania aplikacji IBM MQ przy użyciu wyzwalaczy.

“Praca z interfejsem MQI i klastrami” na stronie 978
Istnieją specjalne opcje dotyczące wywołań i kodów powrotu, które odnoszą się do technologii klastrowej.

“Używanie i zapisywanie aplikacji w systemie IBM MQ for z/OS” na stronie 983
Aplikacje produktu IBM MQ for z/OS mogą być uruchamiane z programów działających w wielu różnych środowiskach. Oznacza to, że mogą skorzystać z udogodnień dostępnych w więcej niż jednym środowisku.

“Aplikacje pomostowe IMS i IMS w systemie IBM MQ for z/OS” na stronie 70
Te informacje ułatwiają pisanie aplikacji produktu IMS przy użyciu produktu IBM MQ.

Otwieranie obiektów za pomocą wywołania MQOPEN

Te informacje umożliwiają zapoznanie się z otwarciem obiektów za pomocą wywołania MQOPEN.

Jako dane wejściowe dla 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 zawsze należy używać uchwytu połączenia zwróconego przez wywołanie MQCONN lub MQCONNX.
- Opis obiektu, który ma zostać otwarty, przy użyciu struktury deskryptora obiektu (MQOD).
- Jedna lub więcej opcji, które sterują działaniem wywołania.

Dane wyjściowe komendy MQOPEN są następujące:

- Uchwyt obiektu, który reprezentuje dostęp do obiektu. Użyj tego parametru na wejściu do wszystkich kolejnych wywołań MQI.

- Zmodyfikowaną strukturę deskryptora obiektu, jeśli tworzona jest kolejka dynamiczna (i jest ona obsługiwana na używanej platformie).
- Kod zakończenia.
- Kod przyczyny.

Zasięg uchwytu obiektu

Zakres uchwytu obiektu (Hobj) jest taki sam, jak zasięg uchwytu połączenia (Hconn).

Jest ona pokryta w produkcie “Zasięg MQCONN lub MQCONNX” na stronie 826 i “Połączenia współużytkowane (niezależne od wątku) z produktem MQCONNX” na stronie 830. W niektórych środowiskach istnieją jednak dodatkowe uwagi:

CICS
W programie CICS można używać uchwytu tylko w ramach tego samego zadania CICS , z którego nawiązałeś wywołanie MQOPEN.

Zadania wsadowe IMS i z/OS
W środowiskach IMS i wsadowych można używać uchwytu w ramach tego samego zadania, ale nie w obrębie żadnych podzadań.

Opisy parametrów wywołania MQOPEN są podane w sekcji MQOPEN.

W poniższych sekcjach opisano informacje, które należy podać jako dane wejściowe dla komendy 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 w przypadku użycia wywołania MQOPEN w celu utworzenia kolejki dynamicznej).

Szczegółowe informacje na temat struktury MQOD można znaleźć w sekcji MQOD.

Informacje na temat korzystania ze struktury MQOD dla list dystrybucyjnych zawiera sekcja “Korzystanie ze struktury MQOD” na stronie 856 w sekcji “Lista dystrybucyjna” na stronie 854.

Rozdzielczość nazwy
W jaki sposób wywołanie MQOPEN rozwiązuje nazwy kolejek i menedżerów kolejek.

Uwaga: Alias menedżera kolejek jest definicją kolejki zdalnej bez pola RNAME .

Po otwarciu kolejki produktu IBM MQ wywołanie MQOPEN wykonuje funkcję rozstrzygania nazw w podanej nazwie 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 jest tłumaczone na nazwę kolejki lokalnej lub kolejki transmisji. Jeśli kolejka jest otwierana dla dowolnego typu danych wejściowych, przeglądania lub ustawiania, jest ona tłumaczona na kolejkę lokalną, jeśli istnieje, i nie powiedzie się, jeśli nie ma ona jednego. Jest ona tłumaczona na kolejkę nielokalną tylko wtedy, gdy jest otwarta tylko dla danych wyjściowych, tylko do zapytania, lub tylko do wyjścia i zapytania. Przegląd procesu rozstrzygania nazw można znaleźć w sekcji Tabela 118 na stronie 836 . Nazwa dostarczona w produkcie *ObjectQMgrName* jest tłumaczana *przed* tym, że w *ObjectName*.

Program Tabela 118 na stronie 836 pokazuje również, w jaki sposób można użyć lokalnej definicji kolejki zdalnej w celu zdefiniowania aliasu dla nazwy menedżera kolejek. Pozwala to wybrać, która kolejka transmisji jest używana podczas umieszczania komunikatów w kolejce zdalnej, dzięki czemu można na przykład użyć pojedynczej kolejki transmisji dla komunikatów przeznaczonych dla wielu menedżerów kolejek zdalnych.

Aby użyć poniższej tabeli, należy najpierw odczytać dwie kolumny z lewej strony, pod nagłówkiem **Dane wejściowe do tabeli MQOD**, a następnie wybrać odpowiednią wielkość liter. Następnie należy przeczytać odpowiadający mu wiersz, postępując zgodnie z instrukcjami. Postępując zgodnie z instrukcjami w kolumnach **Rozstrzygnięte nazwy** , można powrócić do kolumn **Wejście do tabeli MQOD** i wstawić

wartości zgodnie z instrukcjami, albo można wyjść z tabeli z dostarczonymi wynikami. Na przykład może być wymagane wprowadzenie danych wejściowych *ObjectName*.

Tabela 118. Rozstrzyganie nazw kolejek podczas korzystania z komendy MQOPEN

Dane wejściowe dla MQOD	**Dane wejściowe dla MQOD**	**Rozwiązane nazwy**	**Rozwiązane nazwy**	**Rozwiązane nazwy**
ObjectQMgrName	*ObjectName*	*ObjectQMgrName*	*ObjectName*	Transmission queue
Pusty lub lokalny menedżer kolejek	Kolejka lokalna bez atrybutu CLUSTER	Lokalny menedżer kolejek	Wejście *ObjectName*	Nie dotyczy (używana kolejka lokalna)
Pusty menedżer kolejek	Kolejka lokalna z atrybutem CLUSTER	Wybrany menedżer kolejek klastra lub wybrany menedżer kolejek klastra wybrany na PUT zarządzania obciążeniem	Wejście *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE i użyta kolejka lokalna SYSTEM.QSG.TRANSMIT.QUEUE (patrz uwaga)
Lokalny menedżer kolejek	Kolejka lokalna z atrybutem CLUSTER	Lokalny menedżer kolejek	Wejście *ObjectName*	Nie dotyczy (używana kolejka lokalna)
Pusty lub lokalny menedżer kolejek	Kolejka modelowa	Lokalny menedżer kolejek	Nazwa generowana	Nie dotyczy (używana kolejka lokalna)
Pusty lub lokalny menedżer kolejek	Kolejka aliasowa z atrybutem CLUSTER lub bez niej	Ponownie wykonaj tłumaczenie nazw z nazwą *ObjectQMgrName* , a w obiekcie definicji kolejki aliasowej wpisz *ObjectName* ustawioną na wartość *BaseQName* . Nie może być tłumaczany na alias lokalnie zdefiniowany, jeśli określony jest parametr *ObjectQMgrName* , ale może on być tłumaczany na alias klastrowy (udostępniany w innych menedżerach kolejek), w którym wartość *ObjectQMgrName* jest pusta.		

Tabela 118. Rozstrzyganie nazw kolejek podczas korzystania z komendy MQOPEN (kontynuacja)

Dane wejściowe dla MQOD	**Dane wejściowe dla MQOD**	**Rozwiązane nazwy**	**Rozwiązane nazwy**	**Rozwiązane nazwy**
Lokalny menedżer kolejek	Kolejka aliasowa z atrybutem CLUSTER	Alias nie może być tłumaczany na kolejkę klastra, która nie jest zdefiniowana lokalnie, lub nie może być kolejką klastra, która ma taką samą wartość *ObjectName* , co alias.		
Pusty menedżer kolejek	Kolejka aliasowa z atrybutem CLUSTER	Alias może zostać rozstrzygany do kolejki klastra o takiej samej nazwie *ObjectName* , co alias.		
Pusty lub lokalny menedżer kolejek	lokalna definicja kolejki zdalnej	Ponownie wykonaj tłumaczenie nazw z parametrem *ObjectQMgrName* ustawionym na wartość *RemoteQMgrName*, a *ObjectName* ustaw na wartość *RemoteQName*. Nie można rozstrzygnąć kolejek zdalnych		Nazwa atrybutu *XmitQName* , jeśli nazwa inna niż blank; w przeciwnym razie *RemoteQMgrName* znajduje się w obiekcie definicji kolejki zdalnej. SYSTEM.QSG.TRANSMIT.QUEUE (patrz uwaga)
Pusty menedżer kolejek	Nie znaleziono zgodnego obiektu lokalnego; znaleziono kolejki klastra	Wybrany menedżer kolejek klastra lub wybrany menedżer kolejek klastra wybrany na PUT zarządzania obciążeniem	Wejście *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE SYSTEM.QSG.TRANSMIT.QUEUE (patrz uwaga)
Pusty lub lokalny menedżer kolejek	Brak zgodnego obiektu lokalnego; nie znaleziono kolejki klastra		Błąd, kolejka nie została znaleziona	Nie dotyczy
Nazwa menedżera kolejek w tej samej grupie współużytkowania kolejki, co lokalny menedżer kolejek	Lokalna kolejka współużytkowana	Lokalny menedżer kolejek	Wejście *ObjectName*	Nie dotyczy
Nazwa lokalnej kolejki transmisji	(Nierozstrzygnięte)	Dane wejściowe *ObjectQMgrName*	Wejście *ObjectName*	Dane wejściowe *ObjectQMgrName* SYSTEM.QSG.TRANSMIT.QUEUE (patrz uwaga)

Tabela 118. Rozstrzyganie nazw kolejek podczas korzystania z komendy MQOPEN (kontynuacja)

Dane wejściowe dla MQOD	**Dane wejściowe dla MQOD**	**Rozwiązane nazwy**	**Rozwiązane nazwy**	**Rozwiązane nazwy**
Definicja aliasu menedżera kolejek (*RemoteQMgrName* może być nazwą lokalnego menedżera kolejek)	(Nie rozstrzygnięto, kolejka zdalna)	Ponownie wykonaj tłumaczenie nazw z nazwą *ObjectQMgrName* ustawioną na *RemoteQMgrName*. Nie może być rozstrzygane do kolejek zdalnych	Wejście *ObjectName*	Nazwa atrybutu *XmitQName* , jeśli nazwa inna niż blank; w przeciwnym razie *RemoteQMgrName* znajduje się w obiekcie definicji kolejki zdalnej. 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	Wejście *ObjectName*	SYSTEM.CLUSTER.TRANSMIT.QUEUE SYSTEM.QSG.TRANSMIT.QUEUE (patrz uwaga)
Menedżer kolejek nie jest nazwą żadnego obiektu lokalnego; nie znaleziono obiektów klastra	(Nierozstrzygnięte)	Dane wejściowe *ObjectQMgrName*	Wejście *ObjectName*	Atrybut *DefXmitQName* menedżera kolejek, w którym obsługiwany jest program *DefXmitQName* . SYSTEM.QSG.TRANSMIT.QUEUE (patrz uwaga)

Uwagi:

1. *BaseQName* to nazwa kolejki podstawowej z definicji kolejki aliasowej.
2. *RemoteQName* to nazwa kolejki zdalnej z lokalnej definicji kolejki zdalnej.
3. *RemoteQMgrName* to nazwa zdalnego menedżera kolejek z lokalnej definicji kolejki zdalnej.
4. *XmitQName* to nazwa kolejki transmisji z lokalnej definicji kolejki zdalnej.
5. Jeśli używane są menedżery kolejek produktu IBM MQ for z/OS , które są częścią grupy współużytkowania kolejek (QSG), nazwa grupy współużytkowania kolejki może być używana zamiast nazwy lokalnego menedżera kolejek w produkcie Tabela 118 na stronie 836.

   Jeśli lokalny menedżer kolejek nie może otworzyć kolejki docelowej lub umieścić komunikat w kolejce, komunikat jest przesyłany do określonej nazwy ObjectQMgr(w kolejkach wewnątrz grupy lub w kanale IBM MQ).
6. W kolumnie *ObjectName* w tabeli CLUSTER odnosi się zarówno do atrybutów CLUSTER, jak i CLUSNL kolejki.
7. SYSTEM.QSG.TRANSMIT.QUEUE jest używana, 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 przypisana została inna kolejka transmisji klastra, SYSTEM.CLUSTER.TRANSMIT.QUEUE może nie być nazwą kolejki transmisji klastra. Więcej informacji na temat wielu kolejek transmisji klastra zawiera sekcja Łączenie w klastry: planowanie konfigurowania kolejek transmisji klastra.
9. W sytuacji, w której menedżer kolejek nie jest nazwą żadnego obiektu lokalnego, znaleziono menedżery kolejek klastra lub alias menedżera kolejek.

Jeśli nazwa menedżera kolejek została podana przy użyciu produktu **ObjectQMgrName**, a istnieje wiele kanałów klastra o różnych nazwach klastrów znanych przez lokalny menedżer kolejek, które mogą osiągnąć ten cel, to dowolny z tych kanałów może zostać użyty do przeniesienia komunikatu, niezależnie od nazwy klastra kolejki docelowej.

Może to być nieoczekiwane, jeśli komunikaty dla tej kolejki były przewidywane tylko w celu wysłania przez kanał o takiej samej nazwie klastra co kolejka.

Jednak **ObjectQMgrName** ma pierwszeństwo w tym przypadku, a równoważenie obciążenia klastra uwzględnia wszystkie kanały, które mogą osiągnąć ten menedżer kolejek, niezależnie od nazwy klastra, w którym się znajdują.

Otwarcie kolejki aliasowej powoduje również otwarcie kolejki podstawowej, do której alias jest tłumaczący, a otwarcie kolejki zdalnej powoduje otwarcie kolejki transmisji. Z tego powodu nie można usunąć ani określonej kolejki, ani kolejki, do której jest ona tłumaczona, gdy druga jest otwarta.

Podczas gdy kolejka aliasowa nie może być przetłumaczana na inną lokalnie zdefiniowaną kolejkę aliasową (współużytkowana w klastrze lub nie), rozstrzyganie do zdalnej zdefiniowanej kolejki aliasowej klastra jest dozwolone i dlatego może zostać określone jako kolejka podstawowa.

Rozstrzygnięta nazwa kolejki i rozstrzygnięta nazwa menedżera kolejek są przechowywane w polach *ResolvedQName* i *ResolvedQMgrName* w MQOD.

Więcej informacji na temat rozstrzygania nazw w rozproszonym środowisku kolejkowania zawiera sekcja Co to jest rozdzielczość nazw kolejek?.

Korzystanie z opcji wywołania MQOPEN

W parametrze **Options** wywołania MQOPEN należy wybrać jedną lub więcej opcji, aby kontrolować dostęp nadawany obiektowi, który jest otwierany. Za pomocą tych opcji można:

- Otwórz kolejkę i określ, że wszystkie komunikaty umieszczone w tej kolejce muszą być kierowane do tej samej instancji.
- Otwórz kolejkę, aby umożliwić umieszczanie na niej komunikatów.
- Otwórz kolejkę, aby umożliwić przeglądanie komunikatów na niej
- Otwórz kolejkę, aby umożliwić usuwanie z niego komunikatów.
- Otwórz obiekt, aby umożliwić sprawdzenie i ustawienie jego atrybutów (ale można ustawić atrybuty tylko kolejek)
- Otwórz temat lub łańcuch tematu, aby opublikować w nim komunikaty.
- Wiązanie informacji kontekstowych z komunikatem
- Nominuj alternatywny identyfikator użytkownika, który ma być używany do sprawdzania zabezpieczeń
- Sterowanie wywołaniem, jeśli menedżer kolejek znajduje się w stanie wygaszania

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 979.

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 dla umieszczania komunikatów
Aby otworzyć kolejkę lub temat w celu umieszczenia na nim komunikatów, należy użyć opcji MQOO_OUTPUT.

Opcja MQOPEN dla przeglądania komunikatów
Aby otworzyć kolejkę, aby możliwe było *przeglądanie* komunikatów na nim, należy użyć wywołania MQOPEN z opcją MQOO_BROWSE.

Spowoduje to utworzenie *kursora przeglądania* używanego przez menedżer kolejek w celu zidentyfikowania następnego komunikatu w kolejce. Więcej informacji na ten temat zawiera sekcja “Przeglądanie komunikatów w kolejce” na stronie 894.

Uwaga:

1. Nie można przeglądać komunikatów w kolejce zdalnej; nie należy otwierać kolejki zdalnej przy użyciu 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 854.
3. Użyj komendy MQOO_CO_OP w połączeniu z opcją MQOO_BROWSE, jeśli korzystasz z przeglądania kooperatywnego. Patrz Opcje .

Opcje MQOPEN służące do usuwania komunikatów
Trzy opcje sterują otwarciem kolejki w celu usunięcia z niej komunikatów.

W dowolnym wywołaniu MQOPEN można używać tylko jednego z nich. Te opcje definiują, czy program ma wyłączny lub współużytkowany dostęp do kolejki. *Wyłączny dostęp* oznacza, że do czasu zamknięcia kolejki, Tylko użytkownik może usunąć z niego komunikaty. Jeśli inny program podejmie próbę otwarcia kolejki w celu usunięcia komunikatów, jego wywołanie MQOPEN nie powiedzie się. *Dostęp współużytkowany* oznacza, że można usunąć więcej niż jeden program komunikaty z kolejki.

Najbardziej zaleconym podejściem jest zaakceptowanie typu dostępu, który był przeznaczony dla kolejki, gdy kolejka została zdefiniowana. Definicja kolejki zaangażowana jest w ustawienie **Shareability** i **DefInputOpenOption** . Aby zaakceptować ten dostęp, należy skorzystać z opcji MQOO_INPUT_AS_Q_DEF. Informacje o tym, w jaki sposób ustawienie tych atrybutów ma wpływ na typ dostępu, który zostanie podany podczas korzystania z tej opcji, można znaleźć w sekcji Tabela 119 na stronie 840 .

Tabela 119. Sposób, w jaki atrybuty kolejki i opcje wywołania MQOPEN mają wpływ na dostęp do kolejek

Kolejka - atrybuty		**Typ dostępu z opcjami MQOPEN**		
Shareability	**DefInputOpenOption**	**AS_Q_DEF**	**Współużytkowane**	**EXCLUSIVE**

Tabela 119. Sposób, w jaki atrybuty kolejki i opcje wywołania MQOPEN mają wpływ na dostęp do kolejek (kontynuacja)				
Kolejka - atrybuty		**Typ dostępu z opcjami MQOPEN**		
Do współużytkowania	Współużytkowane	shared (współużytkowany)	shared (współużytkowany)	wykluczająca
Do współużytkowania	EXCLUSIVE	wykluczająca	shared (współużytkowany)	wykluczająca
NOT_SHAREABLE*	SHARED*	wykluczająca	wykluczająca	wykluczająca
NOT_SHAREABLE	EXCLUSIVE	wykluczająca	wykluczająca	wykluczająca

Uwaga: * Mimo że można zdefiniować kolejkę tak, aby miała ona taką kombinację atrybutów, domyślna opcja otwierania danych wejściowych jest przesłaniana przez atrybut współużytkowalności.

Lub:

- Jeśli wiadomo, że aplikacja może działać poprawnie, nawet jeśli inne programy mogą usunąć komunikaty z kolejki w tym samym czasie, należy użyć opcji MQOO_INPUT_SHARED. Tabela 119 na stronie 840 pokazuje, w jaki sposób, w niektórych przypadkach, użytkownik otrzyma wyłączny dostęp do kolejki, nawet przy użyciu tej opcji.
- Jeśli wiadomo, że aplikacja może działać poprawnie tylko wtedy, gdy inne programy nie usuwają komunikatów z kolejki w tym samym czasie, należy użyć opcji MQOO_INPUT_EXCLUSIVE.

Uwaga:

1. Nie można usunąć komunikatów z kolejki zdalnej. Z tego powodu nie można otworzyć zdalnej kolejki za pomocą żadnej z opcji MQOO_INPUT_ *.
2. Nie można określić tej opcji podczas otwierania listy dystrybucyjnej. Więcej informacji zawiera sekcja "Lista dystrybucyjna" na stronie 854.

Opcje MQOPEN służące do ustawiania i uzyskiwania informacji o atrybutach
Aby otworzyć kolejkę, tak aby można było ustawić jej atrybuty, należy użyć opcji MQOO_SET.

Nie można ustawić atrybutów żadnego innego typu obiektu (patrz "Sprawdzanie i ustawianie atrybutów obiektu" na stronie 943).

Aby otworzyć obiekt, aby można było dowiedzieć się o jego atrybutach, należy użyć opcji MQOO_INQUIRE.

Uwaga: Nie można określić tej opcji podczas otwierania listy dystrybucyjnej.

Opcje MQOPEN związane z kontekstem komunikatu
Aby możliwe było powiązanie informacji kontekstowych z komunikatem podczas umieszczania go w kolejce, należy użyć jednej z opcji kontekstu komunikatu podczas otwierania kolejki.

Opcje umożliwiają rozróżnianie informacji o kontekście, które odnoszą się do *użytkownika* , który wygenerował komunikat, a który odnosi się do *aplikacji* , która zainicjował dany komunikat. Ponadto można wybrać opcję ustawienia informacji kontekstowych podczas umieszczania komunikatu w kolejce lub wybrać opcję automatycznego uruchamiania kontekstowego z innego uchwytu kolejki.

Pojęcia pokrewne
"Kontekst komunikatu" na stronie 47
Informacja *Kontekst komunikatu* umożliwia aplikacji, która pobiera komunikat, w celu uzyskania informacji o inicjatorze komunikatu.

"Sterowanie informacjami o kontekście komunikatu" na stronie 851
W przypadku użycia wywołania MQPUT lub MQPUT1 w celu umieszczenia komunikatu w kolejce można określić, że menedżer kolejek ma dodać pewne domyślne informacje kontekstu do deskryptora

komunikatu. Aplikacje, które mają odpowiedni poziom uprawnień, mogą dodawać dodatkowe informacje kontekstowe. Do sterowania informacjami kontekstowych 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 otwierania tego obiektu. Jeśli użytkownik nie jest autoryzowany, wywołanie nie powiedzie się.

Jednak programy serwerowe mogą chcieć, aby menedżer kolejek sprawdzał autoryzację użytkownika, dla którego pracują, a nie uprawnienia serwera. W tym celu należy użyć opcji MQOO_ALTERNATE_USER_AUTHORITY wywołania MQOPEN i podać alternatywny identyfikator użytkownika w polu *AlternateUserId* struktury MQOD. Zwykle serwer uzyskałby identyfikator użytkownika z informacji kontekstowych w komunikacie, który jest przetwarzany.

z/OS *Opcja MQOPEN dla wygaszania menedżera kolejek*
W przypadku użycia wywołania MQOPEN, gdy menedżer kolejek znajduje się w stanie wygaszania, wywołanie może się nie powieść, w zależności od używanego środowiska.

W środowisku CICS w systemie z/OS, jeśli używany jest wywołanie MQOPEN, gdy menedżer kolejek jest w stanie wygaszania, wywołanie zawsze nie powiedzie się.

W innych środowiskach z/OS , IBM i, Windows oraz w środowiskach systemów UNIX and Linux wywołanie kończy się niepowodzeniem, gdy menedżer kolejek jest wygaszany tylko wtedy, gdy używana jest opcja MQOO_FAIL_IF_QUIESCING wywołania MQOPEN.

Opcja MQOPEN dla rozstrzygania nazw kolejek lokalnych
Po otwarciu 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 zapełniane nazwami kolejek zdalnych i zdalnego menedżera kolejek, które znajdują się w definicji kolejki zdalnej, lub w wybranej zdalnej kolejce klastra.

Użyj opcji MQOO_RESOLVE_LOCAL_Q w wywołaniu MQOPEN, aby wypełnić *ResolvedQName* w strukturze MQOD nazwą kolejki lokalnej, która została otwarta. *ResolvedQMgrName* jest podobnie wypełniany nazwą lokalnego menedżera kolejek udostępniającego kolejkę lokalną. To pole jest dostępne tylko dla wersji 3 struktury MQOD. Jeśli struktura jest mniejsza niż wersja 3, parametr MQOO_RESOLVE_LOCAL_Q jest ignorowany, jeśli nie zostanie zwrócony błąd.

Jeśli podczas otwierania zostanie podana wartość MQOO_RESOLVE_LOCAL_Q, na przykład kolejka zdalna, *ResolvedQName* to nazwa kolejki transmisji, do której będą umieszczane komunikaty. *ResolvedQMgrName* to nazwa lokalnego menedżera kolejek udostępniającego kolejkę transmisji.

Tworzenie kolejek dynamicznych
Kolejki dynamicznej należy używać, gdy nie jest potrzebna kolejka po zakończeniu aplikacji.

Na przykład można użyć kolejki dynamicznej w celu uzyskania kolejki odpowiedzi. Nazwę kolejki odpowiedzi należy podać w polu *ReplyToQ* struktury MQMD po umieszczniu komunikatu w kolejce (patrz “Definiowanie komunikatów przy użyciu struktury MQMD” na stronie 846).

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 paneli sterowania operacjami i kontrolami. Tworzona kolejka dynamiczna przyjmuje atrybuty kolejki modelowej.

Po wywołaniu komendy MQOPEN należy określić nazwę kolejki modelowej w polu *ObjectName* struktury MQOD. Po zakończeniu wywołania pole *ObjectName* jest ustawione na nazwę utworzonej kolejki dynamicznej. Ponadto pole *ObjectQMgrName* jest ustawione na nazwę lokalnego menedżera kolejek.

Użytkownik może określić nazwę kolejki dynamicznej, która została utworzona na trzy sposoby:

- Podaj pełną nazwę w polu *DynamicQName* struktury MQOD.
- Podaj przedrostek nazwy (mniej niż 33 znaki) i pozwól menedżerowi kolejek na wygenerowanie pozostałej części nazwy. Oznacza to, że menedżer kolejek generuje unikalną nazwę, ale nadal istnieje

kilka elementów sterujących (na przykład użytkownik może chcieć, aby każdy użytkownik mógł użyć określonego przedrostka, lub nadać specjalną klasyfikację zabezpieczeń kolejkom z określonym przedrostkiem w nazwie). Aby użyć tej metody, należy podać gwiazdkę (*) dla ostatniego niepustego znaku w polu *DynamicQName* . Nie należy podawać pojedynczej gwiazdki (*) dla nazwy kolejki dynamicznej.

- Należy zezwolić menedżerowi kolejek na wygenerowanie pełnej nazwy. Aby użyć tej metody, należy podać gwiazdkę (*) w pierwszej pozycji znaku w polu *DynamicQName* .

Więcej informacji na temat tych metod znajduje się w opisie pola DynamicQName .

Więcej informacji na temat kolejek dynamicznych zawiera sekcja Kolejki dynamiczne i modelowe.

Otwieranie kolejek zdalnych

Kolejka zdalna jest kolejką, której właścicielem jest menedżer kolejek inny niż ten, do którego aplikacja jest połączona.

Aby otworzyć zdalną kolejkę, należy użyć wywołania MQOPEN jako kolejki lokalnej. Nazwę kolejki można określić w następujący sposób:

1. W polu *ObjectName* struktury MQOD określ nazwę kolejki zdalnej, która jest znana jako *lokalny* menedżer kolejek.

   Uwaga: W tym przypadku pozostaw puste pole *ObjectQMgrName* .
2. W polu *ObjectName* struktury MQOD określ nazwę kolejki zdalnej, która jest znana jako *zdalny* menedżer kolejek. W polu *ObjectQMgrName* podaj jedną z następujących opcji:

   - Nazwa kolejki transmisji, która ma taką samą nazwę jak zdalny menedżer kolejek. Nazwa i wielkość liter (wielkie, małe lub mieszaninę) muszą być zgodne z *dokładnie*.
   - Nazwa obiektu aliasu menedżera kolejek, który jest tłumaczona na docelowy menedżer kolejek lub kolejkę transmisji.

   Powoduje to, że menedżer kolejek jest miejscem docelowym komunikatu, a także kolejką transmisji, którą należy umieścić w celu uzyskania informacji o tym, czy ma być ona w tym miejscu.
3. Jeśli program *DefXmitQname* jest obsługiwany, w polu *ObjectName* struktury MQOD należy określić nazwę kolejki zdalnej, która jest znana przez *zdalny* menedżer kolejek.

   Uwaga: W polu *ObjectQMgrName* należy ustawić nazwę zdalnego menedżera kolejek (w tym przypadku nie może być ona pusta).

Podczas wywoływania komendy MQOPEN sprawdzana jest poprawność tylko nazw lokalnych; ostatnia operacja sprawdzania oznacza, że istnieje kolejka transmisji, która ma być używana.

Te metody są podsumowane w produkcieTabela 118 na stronie 836.

Zamykanie obiektów przy użyciu wywołania MQCLOSE

Aby zamknąć obiekt, należy użyć wywołania MQCLOSE.

Jeśli obiekt jest kolejką, należy zwrócić uwagę na następujące informacje:

- Przed zamkniętą kolejką dynamiczną nie ma potrzeby tworzenia pustej kolejki dynamicznej.

  Po zamknięciu tymczasowej kolejki dynamicznej kolejka jest usuwana wraz z komunikatami, które mogą nadal być w niej dostępne. Jest to prawda, nawet jeśli nie istnieją niezatwierdzone wywołania MQGET, MQPUT lub MQPUT1 dla kolejki.
- W systemie IBM MQ for z/OS, jeśli istnieją żądania MQGET z opcją MQGMO_SET_SIGNAL dla tej kolejki, są one anulowane.
- Jeśli kolejka została otwarta za pomocą opcji MQOO_BROWSE, kursor przeglądania zostanie zniszczony.

Zamknięcie nie jest powiązane z punktem synchronizacji, dlatego można zamknąć kolejki przed lub po punkcie synchronizacji.

Jako dane wejściowe dla wywołania MQCLOSE należy podać:

- Uchwyt połączenia. Użyj tego samego uchwytu połączenia, który został użyty do jego otwarcia, lub dla aplikacji CICS w systemie z/OS, można 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 zamykano stałą kolejkę dynamiczną).
- Opcja sterująca, aby określić, czy menedżer kolejek powinien usunąć kolejkę, nawet jeśli nadal istnieją na niej komunikaty (przy zamykaniu trwałej kolejki dynamicznej).

Dane wyjściowe komendy MQCLOSE są następujące:

- Kod zakończenia
- Kod przyczyny
- Uchwyt obiektu, zresetuj do wartości MQHO_UNUSABLE_HOBJ

Opisy parametrów wywołania MQCLOSE są podane w tabeli MQCLOSE.

Umieszczanie komunikatów w kolejce

Te informacje umożliwiają zapoznanie się z informacjami na temat umieszczania komunikatów w kolejce.

Użyj wywołania MQPUT, aby umieścić komunikaty w kolejce. Za pomocą komendy MQPUT można wielokrotnie umieszczać wiele komunikatów w tej samej kolejce, po początkowym wywołaniu MQOPEN. Po zakończeniu umieszczania wszystkich komunikatów w kolejce należy wywołać komendę MQCLOSE.

Jeśli chcesz umieścić pojedynczy komunikat w kolejce i natychmiast zamknąć kolejkę, można użyć wywołania MQPUT1 . MQPUT1 wykonuje te same funkcje, co w przypadku następującej sekwencji wywołań:

- MQOPEN
- MQPUT
- MQCLOSE

Jeśli jednak istnieje więcej niż jeden komunikat do umieszczenia w kolejce, jest on bardziej wydajny, aby można było używać wywołania MQPUT. Zależy to od wielkości komunikatu i platformy, nad którą pracuje użytkownik.

Aby dowiedzieć się więcej na temat umieszczania komunikatów w kolejce, należy użyć następujących odsyłaczy:

- “Umieszczanie komunikatów w kolejce lokalnej przy użyciu wywołania MQPUT” na stronie 845
- “Umieszczanie komunikatów w kolejce zdalnej” na stronie 850
- “Ustawianie właściwości komunikatu” na stronie 850
- “Sterowanie informacjami o kontekście komunikatu” na stronie 851
- “Umieszczanie jednego komunikatu w kolejce przy użyciu wywołania MQPUT1” na stronie 853
- “Lista dystrybucyjna” na stronie 854
- “Niektóre przypadki, w których wywołania put nie powiodły się” na stronie 859

Pojęcia pokrewne

“Interfejs kolejki komunikatów-przegląd” na stronie 811
Informacje na temat komponentów interfejsu kolejek komunikatów (Message Queue Interface-MQI).

“Nawiąże połączenie z menedżerem kolejek i odłączenie go od niego” na stronie 824
Aby można było korzystać z usług programistycznych produktu IBM MQ , program musi mieć połączenie z menedżerem kolejek. Informacje zawarte w tej sekcji umożliwiają zapoznanie się z informacjami na temat nawiązywania połączenia z menedżerem kolejek i z jego rozłączeniem.

“Otwieranie i zamykanie obiektów” na stronie 833
Te informacje udostępniają wgląd w otwieranie i zamykanie obiektów produktu IBM MQ .

“Pobieranie komunikatów z kolejki” na stronie 860

Ta sekcja zawiera informacje na temat pobierania komunikatów z kolejki.

“Sprawdzanie i ustawianie atrybutów obiektu” na stronie 943
Atrybuty to właściwości, które definiują parametry obiektu IBM MQ .

“Zatwierdzanie i wycofywanie jednostek pracy” na stronie 946
W tej sekcji opisano sposób zatwierdzania i tworzenia kopii zapasowych wszystkich możliwych do odtworzenia operacji get i put, które wystąpiły w jednostce pracy.

“Uruchamianie aplikacji produktu IBM MQ przy użyciu wyzwalaczy” na stronie 958
Informacje na temat wyzwalaczy i sposobów uruchamiania aplikacji IBM MQ przy użyciu wyzwalaczy.

“Praca z interfejsem MQI i klastrami” na stronie 978
Istnieją specjalne opcje dotyczące wywołań i kodów powrotu, które odnoszą się do technologii klastrowej.

“Używanie i zapisywanie aplikacji w systemie IBM MQ for z/OS” na stronie 983
Aplikacje produktu IBM MQ for z/OS mogą być uruchamiane z programów działających w wielu różnych środowiskach. Oznacza to, że mogą skorzystać z udogodnień dostępnych w więcej niż jednym środowisku.

“Aplikacje pomostowe IMS i IMS w systemie IBM MQ for z/OS” na stronie 70
Te informacje ułatwiają pisanie aplikacji produktu IMS przy użyciu produktu IBM MQ.

Umieszczanie komunikatów w kolejce lokalnej przy użyciu wywołania MQPUT

Ta sekcja zawiera informacje na temat umieszczania 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. Ma to postać struktury deskryptora komunikatu (MQMD).
- Informacje sterujące w postaci struktury opcji put-message (put-message options-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, to zwróci również strukturę opcji i strukturę deskryptora komunikatu. Wywołanie modyfikuje strukturę opcji w celu wyświetlenia nazwy kolejki i menedżera kolejek, do którego komunikat został wysłany. Jeśli użytkownik zażądał, aby menedżer kolejek wygenerował unikalną wartość identyfikatora wprowadzanego komunikatu (przez podanie wartości zero binarnej w polu *MsgId* struktury MQMD), wywołanie wstawia wartość w polu *MsgId* przed zwróceniem tej struktury do użytkownika. Przed wydaniem kolejnej operacji MQPUT należy zresetować tę wartość.

W tabeli MQPUTznajduje się opis wywołania MQPUT.

Więcej informacji na temat informacji wymaganych jako dane wejściowe dla wywołania MQPUT można znaleźć w następujących odsyłaczach:

- “Określanie uchwytów” na stronie 846
- “Definiowanie komunikatów przy użyciu struktury MQMD” na stronie 846
- “Określanie opcji przy użyciu struktury MQPMO” na stronie 846
- “Dane w komunikacie” na stronie 849
- “Umieszczanie komunikatów: korzystanie z uchwytów komunikatów” na stronie 850

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 uchwyt połączenia zwracany przez wywołanie MQCONN lub MQCONNX. W przypadku innych aplikacji zawsze należy używać uchwytu połączenia zwróconego przez wywołanie MQCONN lub MQCONNX.

Niezależnie od środowiska, w którym pracuje użytkownik, 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 wejściowym/wyjściowym dla wywołań MQPUT i MQPUT1 . Użyj go do zdefiniowania komunikatu umieszczanego w kolejce.

Jeśli wartość MQPRI_PRIORITY_AS_Q_DEF lub MQPER_PERSISTENCE_AS_Q_DEF została określona dla komunikatu, a kolejka jest kolejką klastra, użyte wartości są wartościami kolejki, do której komenda MQPUT jest tłumaczona. Jeśli ta kolejka jest wyłączona dla operacji MQPUT, wywołanie nie powiedzie się. Więcej informacji na ten temat zawiera sekcja Konfigurowanie klastra menedżera kolejek .

Uwaga: Przed umieszczniem nowego komunikatu należy użyć wartości MQPMO_NEW_MSG_ID i MQPMO_NEW_CORREL_ID, aby upewnić się, że *MsgId* i *CorrelId* są unikalne. Wartości w tych polach są zwracane w przypadku pomyślnej operacji MQPUT.

Istnieje wprowadzenie do właściwości komunikatu, które zawiera opis MQMD w produkcie “Komunikaty produktu IBM MQ” na stronie 17, a także opis samej struktury w produkcie MQMD.

Określanie opcji przy użyciu struktury MQPMO

Struktura MQPMO (Put Message Option) służy do przekazywania opcji do wywołań MQPUT i MQPUT1 .

Poniższe sekcje dają pomoc przy wypełnianiu pól tej struktury. Opis struktury znajduje się 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 komunikatów put. Jest to 4-znakowe pole. Zawsze należy podać identyfikator MQPMO_STRUC_ID.

Wersja
W tym temacie opisano 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 854). Jeśli zostanie wprowadzona wartość MQPMO_VERSION_3, można

użyć uchwytów komunikatów i właściwości komunikatów. Jeśli zostanie wprowadzona wartość MQPMO_CURRENT_VERSION, aplikacja jest ustawiana zawsze w celu użycia najnowszej wersji.

Opcje
Steruje on następującymi elementami:

- Informacja o tym, czy operacja put jest uwzględniona w jednostce pracy
- Informacje o tym, ile informacji kontekstowych jest powiązanych z komunikatem
- Miejsce, z którego pochodzą informacje o kontekście
- Określa, czy wywołanie nie powiedzie się, jeśli menedżer kolejek jest w stanie wygaszania.
- Określa, czy grupowanie lub segmentacja jest dozwolona
- Generowanie nowego identyfikatora komunikatu i identyfikatora korelacji
- Kolejność umieszczania komunikatów i segmentów w kolejce.
- Określenie, czy należy rozstrzygnąć nazwy kolejek lokalnych

Jeśli pole *Options* zostanie pozostaw ustawione na wartość domyślną (MQPMO_NONE), zostanie wyświetlony komunikat zawierający domyślne informacje o kontekście.

Ponadto sposób, w jaki połączenie działa z punktami synchronizacji, jest określany przez platformę. Wartością domyślną elementu sterującego punktu synchronizacji jest yes (tak) w produkcie z/OS ; dla innych platform, to nie jest.

Kontekst
Określa nazwę uchwytu kolejki, z którego mają być kopiowane informacje kontekstowe (jeśli jest to wymagane w polu *Options*).

Informacje na temat wprowadzenia do kontekstu komunikatów zawiera sekcja “Kontekst komunikatu” na stronie 47. Informacje na temat używania struktury MQPMO w celu sterowania informacjami o kontekście w komunikacie zawiera sekcja “Sterowanie informacjami o kontekście komunikatu” na stronie 851.

ResolvedQName
Nazwa ta zawiera nazwę (po rozstrzygnięcie dowolnej nazwy aliasu) kolejki, która została otwarta w celu odebrania komunikatu. To jest pole wyjściowe.

Nazwa ResolvedQMgr
Nazwa ta zawiera nazwę (po rozstrzygnięcie dowolnej nazwy aliasu) menedżera kolejek, który jest właścicielem kolejki w programie *ResolvedQName*. To jest pole wyjściowe.

Zmaterializowana tabela zapytania (MQPMO) może również zawierać pola wymagane dla list dystrybucyjnych (patrz “Lista dystrybucyjna” na stronie 854). Jeśli ma być używany ten obiekt, używana jest wersja 2 struktury MQPMO. Obejmuje to następujące pola:

RecsPresent
To pole zawiera liczbę kolejek znajdujących się na liście dystrybucyjnej, to znaczy liczbę obecnych rekordów operacji umieszczania komunikatów (MQPMR) i odpowiadających im rekordów odpowiedzi (MQRR).

Wprowadzona wartość może być taka sama, jak liczba rekordów obiektów udostępnionych w tabeli MQOPEN. Jeśli jednak wartość jest mniejsza niż liczba rekordów obiektów udostępnionych w wywołaniu MQOPEN lub jeśli użytkownik nie udostępni żadnych rekordów umieszczania komunikatów, wartości tych kolejek, które nie są zdefiniowane, są pobierane z wartości domyślnych udostępnianych przez deskryptor komunikatu. Ponadto, jeśli wartość jest większa niż liczba udostępnionych rekordów obiektów, nadmiarowe rekordy umieszczania komunikatów są ignorowane.

Zaleca się, aby wykonać jedną z następujących czynności:

- Jeśli chcesz otrzymywać raport lub odpowiedź z każdego miejsca docelowego, wprowadź tę samą wartość, która jest wyświetlana w strukturze MQOR i użyj pól MQPMRs zawierających pola *MsgId* . Zainicjuj te pola *MsgId* na zero lub podaj MQPMO_NEW_MSG_ID.

Po umieszczniu komunikatu w kolejce wartości *MsgId* utworzone przez menedżer kolejek stają się dostępne w produkcie MQPMRs;, aby określić, które miejsce docelowe jest powiązane z każdym raportem lub odpowiadając na odpowiedź.

- Jeśli nie chcesz otrzymywać raportów lub odpowiedzi, wybierz jedną z następujących opcji:

  1. Aby zidentyfikować miejsca docelowe, które nie powiodły się natychmiast, można w dalszym ciągu wprowadzić tę samą wartość w polu *RecsPresent* , co jest wyświetlane w strukturze MQOR i zapewnić produktom MQRRs identyfikowanie tych miejsc docelowych. Nie należy określać żadnych obiektów MQPMRs.
  2. Jeśli nie chcesz określać miejsc docelowych, które nie powiodły się, wprowadź wartość zero w polu *RecsPresent* i nie dostarczaj tabel MQPMR ani MQRRs.

Uwaga: Jeśli używana jest wartość MQPUT1, liczba wskaźników rekordów odpowiedzi i przesunięć rekordów odpowiedzi musi wynosić zero.

Pełny opis rekordów Put Message Records (MQPMR) i Response Records (MQRR) można znaleźć w sekcji MQPMR i MQRR.

PutMsgRecFields
Wskazuje to, które pola są obecne w każdym rekordzie komunikatu umieszczania komunikatów (MQPMR). Lista tych pól znajduje się w sekcji “Korzystanie ze struktury MQPMR” na stronie 858.

PutMsgRecOffset* i *PutMsgRecPtr
Wskaźniki (zwykle w języku C) i offsety (zwykle w języku COBOL) są używane do adresowania rekordów umieszczania komunikatów (patrz sekcja “Korzystanie ze struktury MQPMR” na stronie 858 , aby uzyskać przegląd struktury MQPMR).

Użyj pola *PutMsgRecPtr* , aby określić wskaźnik do pierwszego rekordu umieszczonego komunikatu, lub pole *PutMsgRecOffset* , aby określić przesunięcie pierwszego rekordu umieszczania komunikatu. Jest to przesunięcie od początku wywołania 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 użyć rekordów odpowiedzi (więcej informacji na temat rekordów odpowiedzi można znaleźć w sekcji “Korzystanie ze struktury MQRR” na stronie 857).

Użyj pola *ResponseRecPtr* , aby określić wskaźnik do pierwszego rekordu odpowiedzi, lub pole *ResponseRecOffset* , aby określić przesunięcie pierwszego rekordu odpowiedzi. Jest to przesunięcie od początku struktury MQPMO. Wprowadź wartość inną niż NULL dla opcji *ResponseRecOffset* lub *ResponseRecPtr*.

Uwaga: Jeśli do umieszczania komunikatów na liście dystrybucyjnej używany jest parametr MQPUT1 , wartość *ResponseRecPtr* musi mieć wartość NULL lub wartość zero, a wartość *ResponseRecOffset* musi wynosić zero.

W wersji 3 struktury MQPMO dodatkowo znajdują się następujące pola:

Uchwyt komunikatu OriginalMsg
Użycie tego pola może być uzależnione od wartości pola *Działanie* . W przypadku umieszczania nowego komunikatu z powiązanymi właściwościami komunikatu należy ustawić to pole na uchwyt komunikatu, który został wcześniej utworzony, i ustawić właściwości na. W przypadku przekazywania, odpowiadania na raport lub generowania raportu w odpowiedzi na wcześniej pobrany komunikat, pole to zawiera uchwyt komunikatu tego komunikatu.

Uchwyt NewMsg
Jeśli zostanie określony uchwyt *NewMsg(Nowy komunikat)*, wszystkie właściwości powiązane z właściwościami nadpisania uchwytu są powiązane z uchwytem *OriginalMsgHandle*. Więcej informacji na ten temat zawiera sekcja Działanie (MQLONG).

Działanie
To pole służy do określania typu wykonywanego zadania. Możliwe wartości i ich znaczenia są następujące:

MQACTP_NEW
Jest to nowy komunikat niezwiązany z żadnym innym.

MQACTP_FORWARD
Ten komunikat został pobrany wcześniej i jest teraz przekazywany.

ODPOWIEDŹ MQACTP_REPLY
Ten komunikat jest odpowiedzią na wcześniej 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 w celu określenia, które subskrypcje mają być odbierane. Ta publikacja otrzyma tylko subskrypcje o wartości *SubLevel* mniejszej lub równej tej wartości. Wartością domyślną jest 9, która jest najwyższym poziomem i oznacza, że subskrypcje z dowolnym *SubLevel* mogą otrzymać tę publikację.

Dane w komunikacie

Należy podać adres buforu, który zawiera dane w parametrze **Buffer** wywołania MQPUT.
W komunikatach można umieścić dowolne dane w komunikatach. 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 jest wstawiany komunikat.
- Wielkość dowolnego nagłówka komunikatu dodanego przez program IBM MQ (w tym nagłówek niedostarczonych komunikatów, MQDLH i nagłówek listy dystrybucyjnej, MQDH)

Atrybut **MaxMsgLength** menedżera kolejek przechowuje wielkość komunikatu, który może być przetwarzany przez menedżer kolejek. Wartość domyślna to 100 MB dla wszystkich produktów IBM MQ w wersji V6 lub nowszej.

Aby określić wartość tego atrybutu, należy użyć wywołania MQINQ w obiekcie menedżera kolejek.
W przypadku dużych komunikatów można zmienić tę wartość.

Atrybut **MaxMsgLength** kolejki określa maksymalną wielkość komunikatu, który można umieścić w kolejce. W przypadku próby umieszczenia komunikatu o wielkości większej niż wartość tego atrybutu wywołanie MQPUT nie powiedzie się. Jeśli komunikat jest umieszczany w kolejce zdalnej, maksymalna wielkość komunikatu, którą można pomyślnie umieścić, jest określana przez atrybut **MaxMsgLength** kolejki zdalnej, wszystkich pośrednich kolejek transmisji, które komunikat jest umieszczany na trasie, do miejsca docelowego i używanych kanałów.

W przypadku operacji MQPUT wielkość komunikatu musi być mniejsza lub równa atrybutowi **MaxMsgLength** zarówno w kolejce, jak i w menedżerze kolejek. Wartości tych atrybutów są niezależne, ale zalecane jest ustawienie *MaxMsgLength* kolejki na wartość mniejszą lub równą wartości tego menedżera kolejek.

Program IBM MQ dodaje informacje nagłówka do komunikatów w następujących okolicznościach:

- Po umieszczniu komunikatu w kolejce zdalnej program IBM MQ dodaje do komunikatu strukturę nagłówka transmisji (MQXQH). Ta struktura obejmuje nazwę kolejki docelowej i jej menedżera kolejek, do którego należy.
- Jeśli program IBM MQ nie może dostarczyć komunikatu do kolejki zdalnej, próbuje on umieścić komunikat w kolejce niedostarczonych komunikatów (niedostarczonych komunikatów). Dodaje ona strukturę MQDLH do komunikatu. Struktura ta obejmuje nazwę kolejki docelowej oraz przyczynę umieszczenia komunikatu w kolejce niedostarczonych komunikatów.
- Jeśli komunikat ma zostać wysłany do wielu kolejek docelowych, program IBM MQ dodaje do komunikatu nagłówek MQDH. Opisuje dane znajdujące się w komunikacie, należącym do listy

dystrybucyjnej, w kolejce transmisji. Należy wziąć pod uwagę to, wybierając optymalną wartość dla maksymalnej długości komunikatu.

- Jeśli komunikat jest segmentem lub komunikatem w grupie, produkt IBM MQ może dodać produkt MQMDE.

Struktury te są opisane w tabelach MQDH i MQMDE.

Jeśli komunikaty mają maksymalną wielkość dozwoloną dla tych kolejek, dodanie tych nagłówków oznacza, że operacje put nie powiodą się, ponieważ komunikaty są teraz zbyt duże. Aby zmniejszyć prawdopodobieństwo niepowodzenia operacji put:

- Wielkość komunikatów musi być mniejsza niż wartość atrybutu **MaxMsgLength** kolejek transmisji i niedostarczonych komunikatów. Zezwól co najmniej na wartość stałej MQ_MSG_HEADER_LENGTH (więcej dla dużych list dystrybucyjnych).
- Upewnij się, że atrybut **MaxMsgLength** w kolejce niedostarczonych komunikatów jest ustawiony na wartość taką samą, jak *MaxMsgLength* menedżera kolejek, który jest właścicielem kolejki niedostarczonych komunikatów.

Atrybuty dla menedżera kolejek i stałe kolejkowania komunikatów są opisane w sekcji Atrybuty dla menedżera kolejek.

z/OS Informacje na temat sposobu obsługi niedostarczanych komunikatów w rozproszonym środowisku kolejkowania można znaleźć w sekcji Niedostarczone/nieprzetworzone komunikaty.

Umieszczanie komunikatów: korzystanie z uchwytów komunikatów

Dwa uchwyty komunikatów są dostępne w strukturze MQPMO, *Uchwyt OriginalMsg* i *Uchwyt NewMsg*. Relacja między tymi uchwytami komunikatów jest definiowana przez wartość w polu *Działanie* MQPMO.

Szczegółowe informacje na ten temat zawiera sekcja Działanie (MQLONG). Uchwyt komunikatu nie musi być wymagany w celu umieszczenia komunikatu. Jego celem jest powiązanie właściwości z komunikatem, więc jest on wymagany tylko wtedy, gdy używane są właściwości komunikatu.

Umieszczanie komunikatów w kolejce zdalnej

Aby umieścić komunikat w kolejce zdalnej (czyli w kolejce należnej do menedżera kolejek innego niż ten, do którego aplikacja jest podłączona), a nie do kolejki lokalnej, jedynym dodatkowym zagadnieniem jest sposób określania nazwy kolejki po jej otwarciu.Jest to opisane w sekcji “Otwieranie kolejek zdalnych” na stronie 843. Nie ma żadnych 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

Dla każdej właściwości, która ma zostać ustawiona, wywołaj komendę MQSETMP. Po umieszczonym w komunikacie uchwycie komunikatu i polach działania struktury MQPMO.

Aby powiązać właściwości z komunikatem, komunikat musi mieć uchwyt komunikatu. Utwórz uchwyt komunikatu przy użyciu wywołania funkcji MQCRTMH. Wywołaj komendę MQSETMP, podając ten uchwyt komunikatu dla każdej właściwości, która ma zostać ustawiona. Dostępny jest przykładowy program amqsstma.cilustrujący użycie komendy MQSETMP.

Jeśli jest to nowy komunikat, po umieszczniu go w kolejce przy użyciu komendy MQPUT lub MQPUT1należy ustawić pole uchwytu OriginalMsgw MQPMO na wartość tego uchwytu komunikatu, a następnie ustawić pole MQPMO Action na wartość MQACTP_NEW (jest to wartość domyślna).

Jeśli jest to komunikat, który został wcześniej pobrany, a następnie przekazujesz lub odpowiadasz na niego lub wysyłając raport w odpowiedzi na ten komunikat, umieść ten oryginalny uchwyt komunikatu w polu OriginalMsguchwytu MQPMO i nowym uchwycie komunikatu w polu Uchwyt NewMsg. W zależności od potrzeb ustaw pole Działanie odpowiednio na MQACTP_FORWARD, MQACTP_REPLY lub MQACTP_REPORT.

Jeśli użytkownik ma właściwości w nagłówku MQRFH2 z wcześniej pobranego komunikatu, można przekształcić je w właściwości uchwytu komunikatu przy użyciu wywołania MQBUFMH.

Jeśli komunikat jest umieszczany w kolejce menedżera kolejek na poziomie wcześniejszym niż IBM WebSphere MQ 7.0, który nie może przetwarzać właściwości komunikatu, można ustawić parametr PropertyControl w definicji kanału, aby określić sposób, w jaki właściwości mają być traktowane.

Sterowanie informacjami o kontekście komunikatu

W przypadku użycia wywołania MQPUT lub MQPUT1 w celu umieszczenia komunikatu w kolejce można określić, że menedżer kolejek ma dodać pewne domyślne informacje kontekstu do deskryptora komunikatu. Aplikacje, które mają odpowiedni poziom uprawnień, mogą dodawać dodatkowe informacje kontekstowe. Do sterowania informacjami kontekstowych można użyć pola opcji w strukturze MQPMO.

Informacje o kontekście komunikatu pozwalają aplikacji, która pobiera komunikat, aby dowiedzieć się o inicjatorze komunikatu. Wszystkie informacje kontekstowe są zapisywane w polach kontekstu deskryptora komunikatu. Typ informacji jest objęty tożsamością, pochodzeniem i informacjami o kontekście użytkownika.

Aby sterować informacjami o kontekście, należy użyć pola *Options* w strukturze MQPMO.

Jeśli nie zostaną podane żadne opcje dla informacji o kontekście, menedżer kolejek nadpisuje informacje kontekstu, które mogą znajdować się już w deskryptorze komunikatu, wraz z informacjami o tożsamości i kontekście wygenerowanymi dla komunikatu. Jest to takie samo, jak określenie 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 uzyskiwania informacji).

Jeśli z komunikatem nie ma powiązanych informacji o kontekście, należy skorzystać z opcji MQPMO_NO_CONTEXT. Podczas umieszczania komunikatu bez kontekstu wszystkie operacje sprawdzania uprawnień dokonywane przez produkt IBM MQ są wykonywane przy użyciu pustego ID użytkownika. Pusty identyfikator użytkownika nie może być przypisany do jawnego uprawnienia do zasobów IBM MQ , ale jest traktowany jako członek grupy specjalnej 'nobody'. Więcej informacji na temat grupy specjalnej nobodyzawiera sekcja Informacje uzupełniające dotyczące interfejsu usług instalowalnych.

Ustawienie kontekstu można wykonać za pomocą komendy MQOPEN, po której następuje wywołanie MQPUT przy użyciu opcji MQOO_ i opcji MQPMO_, która została wskazana w poniższych sekcjach. Można również ustawić kontekst przy użyciu tylko MQPUT1, w którym to przypadku wystarczy wybrać opcję MQPMO_ wskazaną w poniższych sekcjach.

W poniższych sekcjach tego tematu wyjaśniono sposób użycia kontekstu tożsamości, kontekstu użytkownika i całego kontekstu.

- “Przekazywanie kontekstu tożsamości” na stronie 851
- “Przekazywanie kontekstu użytkownika” na stronie 852
- “Przekazywanie całego kontekstu” na stronie 852
- “Ustawianie kontekstu tożsamości” na stronie 852
- “Ustawianie kontekstu użytkownika” na stronie 853
- “Ustawianie całego kontekstu” na stronie 853

Przekazywanie kontekstu tożsamości

Ogólnie programy powinny przekazywać informacje kontekstu tożsamości z komunikatu do komunikatu wokół aplikacji, dopóki dane nie dotrą do miejsca docelowego.

Programy powinny zmieniać informacje o kontekście pochodzenia za każdym razem, gdy zmieniają one dane. Jednak aplikacje, które chcą zmienić lub ustawić dowolne informacje kontekstowe, muszą mieć odpowiedni poziom uprawnień. Menedżer kolejek sprawdza to uprawnienie, gdy aplikacje otwierają kolejki; muszą mieć uprawnienia do korzystania z odpowiednich opcji kontekstu dla wywołania MQOPEN.

Jeśli aplikacja pobiera komunikat, przetwarza dane z komunikatu, a następnie umieszcza zmienione dane w innym komunikacie (może to być możliwe do przetworzenia przez inną aplikację), aplikacja

musi przekazać informacje kontekstu tożsamości z oryginalnego komunikatu do nowego komunikatu. Użytkownik może zezwolić menedżerowi kolejek na tworzenie informacji o kontekście pochodzenia.

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 wszystkich innych opcji, które są używane z wywołaniem MQOPEN. Należy jednak pamiętać o tym, że nie można zapisać informacji kontekstowych, jeśli tylko użytkownik przegląda komunikat.

Podczas tworzenia drugiego komunikatu:

- Otwórz kolejkę za pomocą opcji MQOO_PASS_IDENTITY_CONTEXT (dodatkowo do opcji MQOO_OUTPUT).
- W polu *Context* struktury opcji put-message należy podać uchwyt kolejki, z której zostały zapisane informacje o kontekście.
- W polu *Options* w strukturze opcji put-message określ opcję MQPMO_PASS_IDENTITY_CONTEXT.

Przekazywanie kontekstu użytkownika

Nie można wybrać tylko do przekazania kontekstu użytkownika. Aby przekazać kontekst użytkownika podczas umieszczania komunikatu, należy podać parametr MQPMO_PASS_ALL_CONTEXT. Wszystkie właściwości w kontekście użytkownika są przekazywane w taki sam sposób, jak kontekst źródłowy.

Gdy trwa operacja MQPUT lub MQPUT1 , a kontekst jest przekazywany, wszystkie właściwości w kontekście użytkownika są przekazywane z pobranego komunikatu do komunikatu umieszczonego w komunikacie. Wszystkie właściwości kontekstu użytkownika, które zostały zmodyfikowane przez aplikację, są umieszczane razem z ich oryginalnymi wartościami. Wszystkie właściwości kontekstu użytkownika, które zostały usunięte przez umieszczanie aplikacji, są odtwarzane w komunikacie umieszczonym. Wszystkie właściwości kontekstu użytkownika, które aplikacja dodała do wiadomości, są zachowywane.

Przekazywanie całego kontekstu

Jeśli aplikacja pobiera komunikat i umieszcza dane komunikatu (bez zmian) w innym komunikacie, aplikacja musi przekazać wszystkie informacje o kontekście (tożsamość, pochodzenie i użytkownika) 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 tym wyjątkiem, że używana jest opcja MQOPEN MQOO_PASS_ALL_CONTEXT, a opcja put-message MQPMO_PASS_ALL_CONTEXT.

Ustawianie kontekstu tożsamości

Jeśli chcesz ustawić informacje o kontekście tożsamości dla komunikatu:

- Otwórz kolejkę przy użyciu opcji MQOO_SET_IDENTITY_CONTEXT.
- Umieść komunikat w kolejce, podając opcję MQPMO_SET_IDENTITY_CONTEXT. W deskryptorze komunikatu określ wymagane informacje o kontekście tożsamości.

Uwaga: W przypadku ustawienia niektórych (ale nie wszystkich) pól kontekstu tożsamości za pomocą opcji MQOO_SET_IDENTITY_CONTEXT i opcji MQPMO_SET_IDENTITY_CONTEXT należy pamiętać, że menedżer kolejek nie ustawia żadnego z pozostałych pól.

Aby zmodyfikować dowolne opcje kontekstu komunikatu, należy mieć odpowiednie autoryzacje do wystawienia połączenia. Na przykład, aby można było używać funkcji MQOO_SET_IDENTITY_CONTEXT lub MQPMO_SET_IDENTITY_CONTEXT, użytkownik musi mieć uprawnienie `+setid` .

Ustawianie kontekstu użytkownika

Aby ustawić właściwość w kontekście użytkownika, należy ustawić pole Kontekst deskryptora właściwości komunikatu (MQPD) na wartość MQPD_USER_CONTEXT podczas wywoływania wywołania MQSETMP.

Aby ustawić właściwość w kontekście użytkownika, nie trzeba mieć żadnych uprawnień specjalnych. Kontekst użytkownika nie ma opcji kontekstu MQOO_SET_ * lub MQPMO_SET_ *.

Ustawianie całego kontekstu

Aby ustawić zarówno tożsamość, jak i informacje o kontekście pochodzenia 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 informacje o tożsamości i pochodzeniu, które są wymagane.

Dla każdego typu ustawienia kontekstu wymagane jest odpowiednie uprawnienie.

Pojęcia pokrewne

"Kontekst komunikatu" na stronie 47
Informacja *Kontekst komunikatu* umożliwia aplikacji, która pobiera komunikat, w celu uzyskania informacji o inicjatorze komunikatu.

Odsyłacze pokrewne

"Opcje MQOPEN związane z kontekstem komunikatu" na stronie 841
Aby możliwe było powiązanie informacji kontekstowych z komunikatem podczas umieszczania go w kolejce, należy użyć jednej z opcji kontekstu komunikatu podczas otwierania kolejki.

Umieszczanie jednego komunikatu w kolejce przy użyciu wywołania MQPUT1

Użyj wywołania MQPUT1 , jeśli chcesz zamknąć kolejkę natychmiast po umieszczonym na nim pojedynczym komunikacie. Na przykład aplikacja serwera może używać wywołania MQPUT1 , gdy wysyła odpowiedź do każdej z różnych kolejek.

Parametr MQPUT1 jest funkcjonalnie równoważny z wywołaniem komendy MQOPEN, po której następuje wywołanie MQPUT, a następnie MQCLOSE. Jedyną różnicą w składni wywołań MQPUT i MQPUT1 jest to, że dla operacji MQPUT określa się uchwyt obiektu, natomiast dla parametru MQPUT1 należy określić strukturę deskryptora obiektu (MQOD) zdefiniowaną w tabeli MQOPEN (patrz sekcja "Identyfikowanie obiektów (struktura MQOD)" na stronie 835). Jest to spowodowane tym, że należy podać informacje dla wywołania MQPUT1 dotyczące kolejki, która ma zostać otwarta, podczas gdy podczas wywoływania operacji MQPUT kolejka musi być już otwarta.

Jako dane wejściowe dla wywołania MQPUT1 należy podać:

- Uchwyt połączenia.
- Opis obiektu, który ma zostać otwarty. Ma to postać struktury deskryptora obiektu (MQOD).
- Opis komunikatu, który ma zostać umieszczony w kolejce. Ma to postać struktury deskryptora komunikatu (MQMD).
- Informacje sterujące w postaci struktury opcji komunikatu put (put-message options structure- 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, to zwróci również strukturę opcji i strukturę deskryptora komunikatu. Wywołanie modyfikuje strukturę opcji w celu wyświetlenia nazwy kolejki i menedżera kolejek, do którego komunikat został wysłany. Jeśli użytkownik zażądał, aby menedżer kolejek

wygenerował unikalną wartość dla identyfikatora wprowadzanego komunikatu (przez podanie wartości zero w polu *MsgId* struktury MQMD), wywołanie wstawia wartość w polu *MsgId* przed zwróceniem tej struktury do użytkownika.

Uwaga: Nie można użyć komendy MQPUT1 z nazwą kolejki modelowej, jednak po otwarciu kolejki modelowej można wydać komendę MQPUT1 do kolejki dynamicznej.

Sześć parametrów wejściowych dla parametru MQPUT1 to:

Hconn
To jest 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 zawsze należy używać uchwytu połączenia zwróconego przez wywołanie MQCONN lub MQCONNX.

ObjDesc
Jest to struktura deskryptora obiektu (MQOD).

W polach *ObjectName* i *ObjectQMgrName* podaj nazwę kolejki, w której ma zostać umieszczony komunikat, oraz nazwę menedżera kolejek, który jest właścicielem tej kolejki.

Pole *DynamicQName* jest ignorowane dla wywołania MQPUT1 , ponieważ nie może korzystać z kolejek modelowych.

Użyj pola *AlternateUserId* , aby nominować alternatywny identyfikator użytkownika, który ma być używany do testowania uprawnień do otwarcia 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 komunikatów typu put (put-message options) (MQPMO). Należy go używać w taki sam sposób, jak w przypadku wywołania MQPUT (patrz sekcja “Określanie opcji przy użyciu struktury MQPMO” na stronie 846).

Gdy pole *Options* jest ustawione na zero, menedżer kolejek używa własnego ID użytkownika podczas wykonywania testów dla uprawnień dostępu do kolejki. Ponadto menedżer kolejek ignoruje dowolny alternatywny identyfikator użytkownika podany w polu *AlternateUserId* struktury MQOD.

BufferLength
Jest to długość wiadomości.

Buffer
Jest to obszar buforu, który zawiera tekst komunikatu.

Jeśli używane są klastry, komenda MQPUT1 działa tak, jakby w efekcie działa MQOO_BIND_NOT_FIXED. Aplikacje muszą używać rozstrzygniętych pól w strukturze MQPMO, a nie struktury MQOD w celu określenia miejsca, w którym komunikat został wysłany. Więcej informacji na ten temat zawiera sekcja Konfigurowanie klastra menedżera kolejek .

W pliku MQPUT1znajduje się opis wywołania MQPUT1 .

Lista dystrybucyjna

Nie jest obsługiwane 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 otworzyć wiele kolejek, a pojedyncze wywołanie MQPUT może następnie umieścić komunikat dla każdej z tych kolejek. Niektóre informacje ogólne ze struktur MQI używanych dla tego procesu 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.

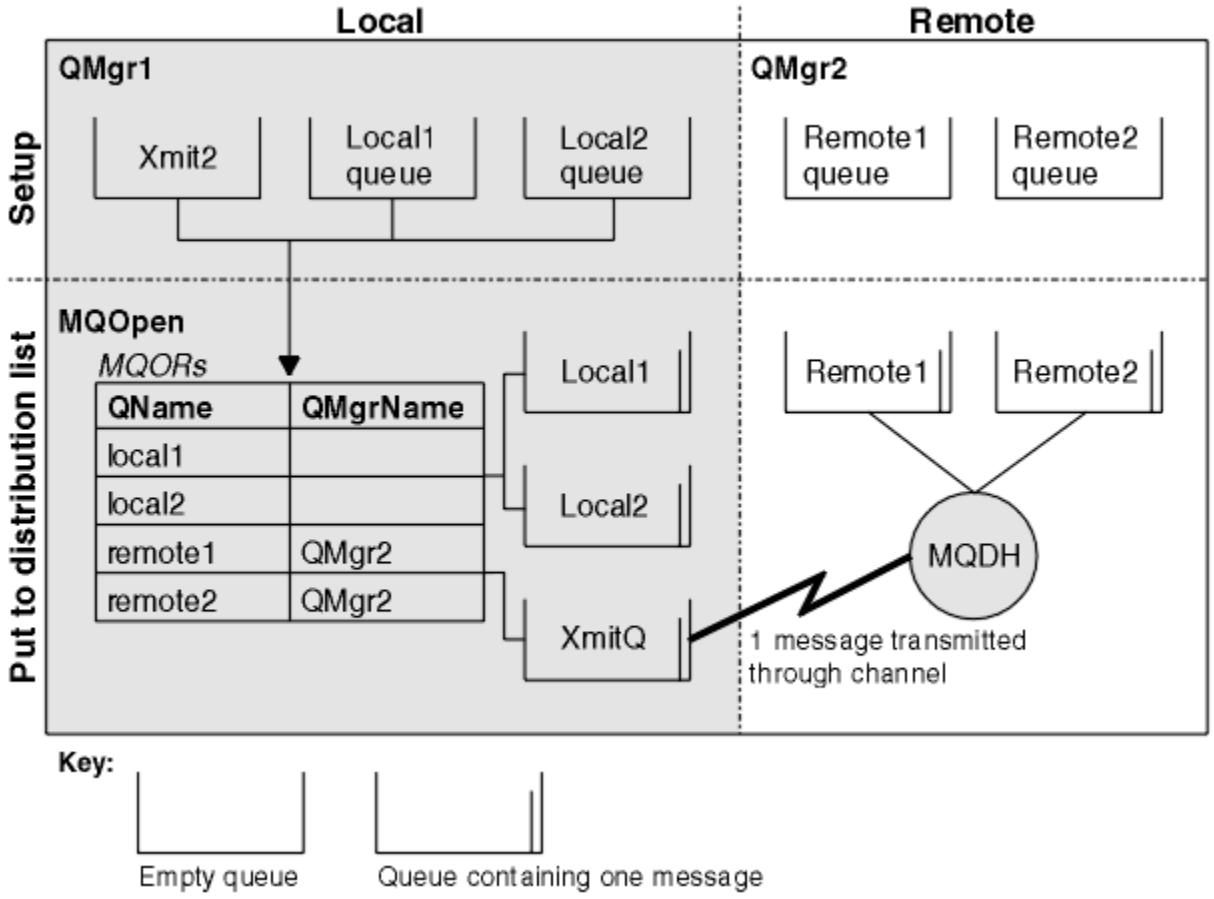
Po wywołaniu wywołania MQOPEN z deskryptora obiektu (MQOD) pobierane są ogólne informacje. Jeśli w polu *Version* zostanie podana wartość MQOD_VERSION_2 , a w polu *RecsPresent* wartość większa od zera, to *Hobj* może zostać zdefiniowany jako uchwyt listy (jednej lub większej liczby kolejek), a nie kolejki. W tym przypadku konkretne informacje są podawane za pośrednictwem rekordów obiektów (MQORs), które nadają szczegółowe informacje na temat miejsca docelowego (to znaczy *ObjectName* i *ObjectQMgrName*).

Uchwyt obiektu (*Hobj*) jest przekazywany do wywołania MQPUT, co pozwala na umieszczenie listy, a nie do pojedynczej kolejki.

Gdy komunikat jest umieszczany w kolejkach (MQPUT), informacje ogólne są pobierane ze struktury opcji umieszczania komunikatów (Put Message Option structure-MQPMO) i deskryptora komunikatu (Message Descriptor-MQMD). Konkretne informacje są podane w postaci rekordów komunikatu umieszczania komunikatów (Put Message Records-MQPMRs).

Rekordy odpowiedzi (MQRR) mogą otrzymać kod zakończenia i kod przyczyny specyficzne dla każdej kolejki docelowej.

Rysunek 61 na stronie 855 pokazuje, w jaki sposób działają listy dystrybucyjne.



Rysunek 61. Sposób działania list dystrybucyjnych

Otwieranie list dystrybucyjnych
Użyj wywołania MQOPEN, aby otworzyć listę dystrybucyjną, a następnie użyj opcji wywołania, aby określić, co ma być używane z listą.

Jako dane wejściowe dla komendy MQOPEN należy podać:

- Uchwyt połączenia (patrz “Umieszczanie komunikatów w kolejce” na stronie 844 , aby uzyskać opis)
- Informacje ogólne w strukturze deskryptora obiektu (MQOD)
- Nazwa każdej kolejki, która ma być otwarta, przy użyciu struktury rekordów obiektów (MQOR)

Dane wyjściowe komendy MQOPEN są następujące:

- Uchwyt obiektu, który reprezentuje 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

Użyj struktury MQOD, aby zidentyfikować kolejki, które mają zostać otwarte.

Aby zdefiniować listę dystrybucyjną, w polu *Version* należy podać wartość MQOD_VERSION_2 , wartość większą od zera w polu *RecsPresent* oraz wartość MQOT_Q w polu *ObjectType* . Sekcja MQOD zawiera opis wszystkich pól struktury MQOD.

Korzystanie ze struktury MQOR

Podaj strukturę MQOR dla każdego miejsca docelowego.

Struktura zawiera nazwy kolejek docelowych i menedżerów kolejek. Pola *ObjectName* i *ObjectQMgrName* w tabeli MQOD nie są używane dla list dystrybucyjnych. Musi istnieć jeden lub więcej rekordów obiektów. Jeśli pole *ObjectQMgrName* pozostanie puste, zostanie użyty lokalny menedżer kolejek. Więcej informacji na temat tych pól można znaleźć w sekcji ObjectName i ObjectQMgrName (Nazwa obiektu ObjectQMgr).

Kolejki docelowe można określić na dwa sposoby:

- Za pomocą pola przesunięcia *ObjectRecOffset*.

  W takim przypadku aplikacja musi zadeklarować własną strukturę zawierającą strukturę MQOD, po której następuje tablica rekordów MQOR (wraz z wymaganą wieloma elementami tablicy), a następnie ustaw *ObjectRecOffset* na przesunięcie pierwszego elementu w tablicy od początku tabeli MQOD. Upewnij się, że przesunięcie jest poprawne.

  Zaleca się korzystanie z wbudowanych urządzeń udostępnianych przez język programowania, jeśli są one dostępne we wszystkich środowiskach, w których działa aplikacja. Poniższy kod ilustruje tę technikę dla języka 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 urządzeń we wszystkich środowiskach, których to dotyczy. Poniższy kod ilustruje tę technikę:

```
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 pomijanie bajtów między tabelą MQOD i tablicą MQOR, należy je dodać do wartości zapisanej w składowaniu *ObjectRecOffset*.

Użycie produktu *ObjectRecOffset* jest zalecane w przypadku języków programowania, które nie obsługują typu danych wskaźnika, lub które implementują typ danych wskaźnika w sposób, który nie jest przenośny dla różnych środowisk (na przykład w języku programowania COBOL).

- Za pomocą pola wskaźnika *ObjectRecPtr*.

  W takim przypadku aplikacja może zadeklarować tablicę struktur MQOR niezależnie od struktury MQOD, a następnie ustawić wartość *ObjectRecPtr* na adres tablicy. Poniższy kod ilustruje tę technikę dla języka programowania C:

```
MQOD MyMqod;
MQOR MyMqor[100];
MyMqod.ObjectRecPtr = MyMqor;
```

  Użycie produktu *ObjectRecPtr* jest zalecane w przypadku języków programowania obsługujących typ danych wskaźnika w sposób przenośny dla różnych środowisk (na przykład język programowania w języku C).

Niezależnie od wybranej techniki, należy użyć jednej z następujących opcji: *ObjectRecOffset* i *ObjectRecPtr* ; wywołanie kończy się niepowodzeniem z kodem przyczyny MQRC_OBJECT_RECORDS_ERROR, jeśli oba są równe zero, albo oba są niezerowe.

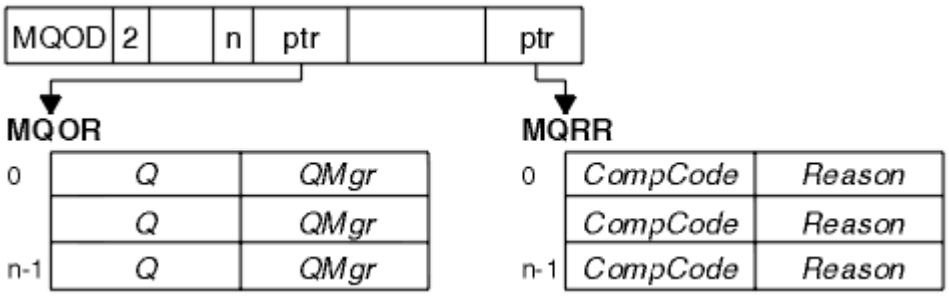
Korzystanie ze struktury MQRR

Te struktury są specyfiką miejsca docelowego; każdy rekord odpowiedzi zawiera pole *CompCode* i *Reason* dla każdej kolejki listy dystrybucyjnej. Tej struktury należy użyć, aby umożliwić rozróżnianie miejsc, w których występują problemy.

Jeśli na przykład zostanie wyświetlony kod przyczyny komendy MQRC_MULTIPLE_REASON, a lista dystrybucyjna zawiera pięć kolejek docelowych, nie będzie wiadomo, do których kolejek mają być stosowane problemy, jeśli ta struktura nie zostanie użyta. Jeśli jednak kod zakończenia i kod przyczyny dla każdego miejsca docelowego są dostępne, można łatwiej znaleźć błędy.

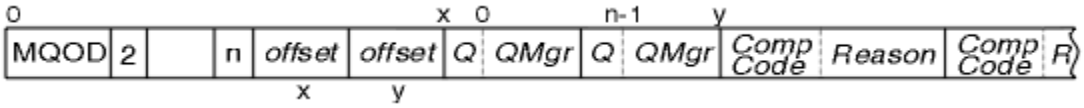
Więcej informacji na temat struktury MQRR zawiera sekcja MQRR .

Rysunek 62 na stronie 857 pokazuje, w jaki sposób można otworzyć listę dystrybucyjną w języku C.



Rysunek 62. Otwieranie listy dystrybucyjnej w języku C

Rysunek 63 na stronie 857 pokazuje, jak można otworzyć listę dystrybucyjną w języku COBOL.



Rysunek 63. Otwieranie listy dystrybucyjnej w języku COBOL

Korzystanie z opcji MQOPEN

Podczas otwierania listy dystrybucyjnej można określić następujące opcje:

- MQOO_OUTPUT
- MQOO_FAIL_IF_QUIESCING (opcjonalnie)

- MQOO_ALTERNATE_USER_AUTHORITY (opcjonalne)
- MQOO_ * _CONTEXT (opcjonalnie)

Opis tych opcji można znaleźć w sekcji "Otwieranie i zamykanie obiektów" na stronie 833 .

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 844).
- Uchwyt obiektu. Jeśli lista dystrybucyjna zostanie otwarta za pomocą komendy MQOPEN, program *Hobj* umożliwia tylko umieszczenie listy dystrybucyjnej na liście.
- Struktura deskryptora komunikatu (MQMD). Opis tej struktury można znaleźć w sekcji MQMD .
- Informacje sterujące w postaci struktury opcji komunikatu put (put-message option structure-MQPMO). Więcej informacji na temat wypełniania pól struktury MQPMO zawiera sekcja "Określanie opcji przy użyciu struktury MQPMO" na stronie 846 .
- Informacje sterujące w postaci rekordów umieszczania komunikatów (Put Message Records-MQPMR).
- Długość danych zawartych w komunikacie (MQLONG).
- Same dane komunikatu.

Dane wyjściowe:

- Kod zakończenia
- Kod przyczyny
- Rekordy odpowiedzi (opcjonalne)

Korzystanie ze struktury MQPMR

Ta struktura jest opcjonalna i udostępnia informacje specyficzne dla miejsca docelowego dla niektórych pól, które mogą być różne od tych, które zostały już zidentyfikowane w deskryptywnym MQMD.

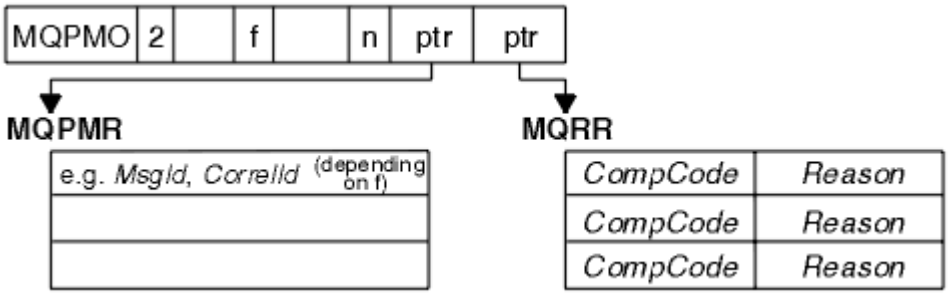
Opis tych pól znajduje się w sekcji MQPMR.

Zawartość każdego rekordu zależy od informacji podanych w polu *PutMsgRecFields* obiektu MQPMO. Na przykład w przykładowym programie AMQSPTL0.C (aby uzyskać opis), w którym pokazano użycie list dystrybucyjnych, w przykładzie "Przykładowy program listy dystrybucyjnej" na stronie 1198 wartości parametrów *MsgId* i *CorrelId* w tabeli MQPMR. Ta sekcja przykładowego programu wygląda następująco:

```
typedef struct
{
MQBYTE24 MsgId;
MQBYTE24 CorrelId;
} PutMsgRec;
...
/**********************
MQLONG PutMsgRecFields=MQPMRF_MSG_ID | MQPMRF_CORREL_ID;
```

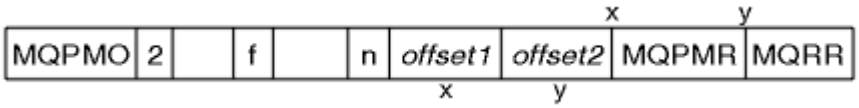
Oznacza to, że *MsgId* i *CorrelId* są dostępne dla każdego miejsca docelowego listy dystrybucyjnej. Rekordy umieszczania komunikatów są udostępniane jako tablica.

Rysunek 64 na stronie 859 pokazuje, w jaki sposób można umieścić komunikat w liście dystrybucyjnej w C.



Rysunek 64. Umieszczanie komunikatu na liście dystrybucyjnej w języku C

Program Rysunek 65 na stronie 859 pokazuje, jak można umieścić komunikat w liście dystrybucyjnej w języku COBOL.



Rysunek 65. Umieszczanie komunikatu na liście dystrybucyjnej w języku COBOL

Korzystanie z komendy MQPUT1

Jeśli używany jest protokół MQPUT1, należy wziąć pod uwagę następujące kwestie:

1. Wartości pól *ResponseRec0ffset* i *ResponseRecPtr* muszą mieć wartość null lub zero.
2. Rekordy odpowiedzi, jeśli jest to wymagane, muszą być adresowane z tabeli MQOD.

Niektóre przypadki, w których wywołania put nie powiodły się

Jeśli niektóre atrybuty kolejki zostaną zmienione za pomocą opcji FORCE w komendzie w okresie między wywołaniem komendy MQOPEN i wywołaniem MQPUT, wywołanie MQPUT nie powiedzie się i zwróci kod przyczyny MQRC_OBJECT_CHANGED.

Menedżer kolejek oznacza, że uchwyt obiektu nie jest już poprawny. Dzieje się tak również wtedy, gdy zmiany są wprowadzane w czasie przetwarzania wywołania MQPUT1 lub gdy zmiany mają zastosowanie do każdej kolejki, do której nazwa kolejki jest tłumaczona. Atrybuty, które mają wpływ na uchwyt w ten sposób, są wymienione w opisie wywołania MQOPEN w produkcie MQOPEN. Jeśli wywołanie zwróci kod przyczyny MQRC_OBJECT_CHANGED, zamknij kolejkę, ponownie go otwórz, a następnie spróbuj ponownie umieścić komunikat.

Jeśli operacje put są zablokowane dla kolejki, w której podjęto próbę umieszczenia komunikatów (lub dowolnej kolejki, do której nazwa kolejki jest tłumaczona), wywołanie MQPUT lub MQPUT1 nie powiedzie się i zwróci kod przyczyny MQRC_PUT_INHIBITED. Może być możliwe pomyślne umieszczenie komunikatu w przypadku podjęcia próby połączenia w późniejszym czasie, jeśli projekt aplikacji jest taki, że inne programy regularnie zmieniają atrybuty kolejek.

Furthemore, jeśli kolejka, na której próbujesz umieścić komunikat, jest pełna, wywołanie MQPUT lub MQPUT1 nie powiedzie się i zwróci wartość MQRC_Q_FULL.

Jeśli usunięto kolejkę dynamiczną (tymczasową lub trwałą), wywołania MQPUT korzystające z wcześniej uzyskanego uchwytu obiektu nie powiodą się i zwróć kod przyczyny MQRC_Q_DELETED. W tej sytuacji dobrą praktyką jest 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 przy użyciu tylko pól wyjściowych *CompCode* i *Reason* w tabelach MQOPEN i MQPUT.

Jeśli listy dystrybucyjne są używane w celu umieszczania komunikatów w wielu miejscach docelowych, rekordy odpowiedzi zawierają określone *CompCode* i *Reason* dla każdego miejsca docelowego. Jeśli zostanie wyświetlony kod zakończenia MQCC_FAILED, żaden komunikat nie zostanie pomyślnie umieszczony w kolejce docelowej. Jeśli kod zakończenia to MQCC_WARNING, to komunikat zostanie

pomyślnie umieszczony w jednej lub większej liczby kolejek docelowych. Jeśli zostanie wyświetlony kod powrotu z MQRC_MULTIPLE_POWODÓW, kody przyczyny nie są takie same dla każdego miejsca docelowego. Dlatego zaleca się użycie struktury MQRR, aby można było określić, która kolejka lub kolejki spowodowały błąd, a także przyczyny 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. Można usunąć komunikat z kolejki, tak aby inne programy nie mogły go już oglądać.
2. Można skopiować wiadomość, pozostawiając oryginalny komunikat w kolejce. Jest to określane jako *przeglądanie*. Można usunąć ten komunikat po jego przeglądowaniu.

W obu przypadkach można użyć wywołania MQGET, ale najpierw aplikacja musi być połączona z menedżerem kolejek i należy użyć wywołania MQOPEN w celu otwarcia kolejki (dla danych wejściowych, przeglądania lub obu tych elementów). Operacje te są opisane w produktach “Nawiąże połączenie z menedżerem kolejek i odłączenie go od niego” na stronie 824 i “Otwieranie i zamykanie obiektów” na stronie 833.

Po otwarciu kolejki można wielokrotnie korzystać z wywołania MQGET w celu przeglądania lub usuwania komunikatów w tej samej kolejce. Wywołaj komendę MQCLOSE po zakończeniu pobierania wszystkich komunikatów, które mają być wyświetlane w kolejce.

Użyj poniższych odsyłaczy, aby dowiedzieć się więcej na temat pobierania komunikatów z kolejki:

- “Pobieranie komunikatów z kolejki przy użyciu wywołania MQGET” na stronie 861
- “Kolejność, w jakiej komunikaty są pobierane z kolejki” na stronie 865
- “Uzyskiwanie określonego komunikatu” na stronie 877
- “Zwiększanie wydajności nietrwałych komunikatów” na stronie 878
- z/OS “Typ indeksu” na stronie 882
- “Obsługa komunikatów o długości większej niż 4 MB” na stronie 883
- “Oczekiwanie na komunikaty” na stronie 889
- z/OS “sygnalizowanie” na stronie 890
- “Pomijanie wycofania” na stronie 891
- “Konwersja danych aplikacji” na stronie 893
- “Przeglądanie komunikatów w kolejce” na stronie 894
- “Niektóre przypadki, w których wywołanie MQGET nie powiodło się” na stronie 900

Pojęcia pokrewne

“Interfejs kolejki komunikatów-przegląd” na stronie 811
Informacje na temat komponentów interfejsu kolejek komunikatów (Message Queue Interface-MQI).

“Nawiąże połączenie z menedżerem kolejek i odłączenie go od niego” na stronie 824
Aby można było korzystać z usług programistycznych produktu IBM MQ , program musi mieć połączenie z menedżerem kolejek. Informacje zawarte w tej sekcji umożliwiają zapoznanie się z informacjami na temat nawiązywania połączenia z menedżerem kolejek i z jego rozłączeniem.

“Otwieranie i zamykanie obiektów” na stronie 833
Te informacje udostępniają wgląd w otwieranie i zamykanie obiektów produktu IBM MQ .

“Umieszczanie komunikatów w kolejce” na stronie 844
Te informacje umożliwiają zapoznanie się z informacjami na temat umieszczania komunikatów w kolejce.

“Sprawdzanie i ustawianie atrybutów obiektu” na stronie 943
Atrybuty to właściwości, które definiują parametry obiektu IBM MQ .

“Zatwierdzanie i wycofywanie jednostek pracy” na stronie 946

W tej sekcji opisano sposób zatwierdzania i tworzenia kopii zapasowych wszystkich możliwych do odtworzenia operacji get i put, które wystąpiły w jednostce pracy.

“Uruchamianie aplikacji produktu IBM MQ przy użyciu wyzwalaczy” na stronie 958
Informacje na temat wyzwalaczy i sposobów uruchamiania aplikacji IBM MQ przy użyciu wyzwalaczy.

“Praca z interfejsem MQI i klastrami” na stronie 978
Istnieją specjalne opcje dotyczące wywołań i kodów powrotu, które odnoszą się do technologii klastrowej.

“Używanie i zapisywanie aplikacji w systemie IBM MQ for z/OS” na stronie 983
Aplikacje produktu IBM MQ for z/OS mogą być uruchamiane z programów działających w wielu różnych środowiskach. Oznacza to, że mogą skorzystać z udogodnień dostępnych w więcej niż jednym środowisku.

“Aplikacje pomostowe IMS i IMS w systemie IBM MQ for z/OS” na stronie 70
Te informacje ułatwiają pisanie aplikacji produktu 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 zwrócony z kolejki. Ma to postać struktury deskryptora komunikatu (MQMD).
- Sterowanie informacjami w postaci struktury MQGMO (Get Message Options).
- Wielkość buforu, który został przypisany do przechowywania komunikatu (MQLONG).
- Adres pamięci masowej, w której ma zostać umieszczony komunikat.

Dane wyjściowe komendy MQGET są następujące:

- Kod przyczyny
- Kod zakończenia
- Komunikat w określonym obszarze buforu, jeśli wywołanie zakończy się pomyślnie.
- Struktura opcji została zmodyfikowana w celu wyświetlenia nazwy kolejki, z której został pobrany komunikat.
- Struktura deskryptora komunikatu wraz z zawartością pól zmodyfikowanych w celu opisania komunikatu, który został pobrany
- Długość komunikatu (MQLONG)

W tabeli MQGETznajduje się opis wywołania MQGET.

W poniższych sekcjach opisano informacje, które należy podać jako dane wejściowe dla wywołania MQGET.

- “Określanie uchwytów połączeń” na stronie 861
- “Opisywanie komunikatów przy użyciu struktury MQMD i wywołania MQGET” na stronie 862
- “Określanie opcji MQGET przy użyciu struktury MQGMO” na stronie 862
- “Określanie wielkości obszaru buforu” na stronie 864

Określanie uchwytów połączeń

z/OS W przypadku aplikacji 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 zawsze należy używać uchwytu połączenia zwróconego przez wywołanie MQCONN lub MQCONNX.

Użyj uchwytu kolejki (*Hobj*), który jest zwracany w przypadku wywołania komendy MQOPEN.

Opisywanie komunikatów przy użyciu struktury MQMD i wywołania MQGET

Aby zidentyfikować komunikat, który ma zostać wygenerowany z kolejki, należy użyć struktury deskryptora komunikatu (MQMD).

Jest to parametr wejściowy/wyjściowy dla wywołania MQGET. Istnieje wprowadzenie do właściwości komunikatu, które zawiera opis MQMD w produkcie “Komunikaty produktu IBM MQ” na stronie 17, a także opis samej struktury w produkcie MQMD.

Jeśli wiadomo, który komunikat ma zostać wygenerowany z kolejki, należy zapoznać się z “Uzyskiwanie określonego komunikatu” na stronie 877.

Jeśli konkretna wiadomość nie zostanie określona, komenda MQGET pobiera komunikat *pierwszy* w kolejce. “Kolejność, w jakiej komunikaty są pobierane z kolejki” na stronie 865 opisuje sposób, w jaki priorytet komunikatu, atrybut **MsgDeliverySequence** kolejki oraz opcja MQGMO_LOGICAL_ORDER określają kolejność komunikatów w kolejce.

Uwaga: Aby użyć komendy MQGET więcej niż raz (na przykład w celu wykonania kroku przez komunikaty w kolejce), po każdym wywołaniu należy ustawić pola *MsgId* i *CorrelId* tej struktury na wartość NULL. Powoduje to wyczyszczenie tych pól identyfikatorów pobranych komunikatów.

Jeśli jednak chcesz zgrupować swoje wiadomości, *GroupId* musi być taka sama dla wiadomości w tej samej grupie, tak aby wywołanie wyszukało komunikat o tych samych identyfikatorach co poprzedni komunikat w celu wykonania całej grupy.

Określanie opcji MQGET przy użyciu struktury MQGMO

Struktura MQGMO jest zmienną wejścia/wyjścia dla przekazywania opcji do wywołania MQGET. W poniższych sekcjach można wykonać niektóre z pól tej struktury.

W produkcie MQGMOznajduje się opis struktury MQGMO.

StrucId
StrucId jest 4-znakowym polem używanym do identyfikowania struktury jako struktury opcji komunikatów get. Zawsze należy podać identyfikator MQGMO_STRUC_ID.

Version
*Version*Opisuje numer wersji struktury. MQGMO_VERSION_1 jest wartością domyślną. Aby użyć pól w wersji 2 lub pobrać komunikaty w kolejności logicznej, należy określić wartość MQGMO_VERSION_2. Aby użyć pól w wersji 3 lub pobrać komunikaty w kolejności logicznej, należy określić wartość MQGMO_VERSION_3. MQGMO_CURRENT_VERSION ustawia aplikację w taki sposób, aby była używana na ostatnim poziomie.

Options
W obrębie kodu można wybrać opcje w dowolnej kolejności; każda opcja jest reprezentowana przez bit w polu *Options* .

Pole *Options* steruje:

- Określa, czy wywołanie MQGET oczekuje na dotarcie komunikatu do kolejki przed jej zakończeniem (patrz “Oczekiwanie na komunikaty” na stronie 889)
- Informacja o tym, czy operacja pobierania jest uwzględniona w jednostce pracy.
- Określa, czy nietrwały komunikat jest pobierany poza punktem synchronizacji, co pozwala na szybkie przesyłanie komunikatów.
- z/OS W systemie IBM MQ for z/OSinformacja o tym, czy pobrany komunikat jest oznaczony jako pominięty (patrz “Pomijanie wycofania” na stronie 891)
- Informacja o tym, czy komunikat jest usuwany z kolejki, czy tylko przeglądane
- Czy wybrać komunikat przy użyciu kursora przeglądania lub innego kryterium wyboru
- Określa, czy wywołanie powiedzie się nawet wtedy, gdy komunikat jest dłuższy niż bufor.
- z/OS W systemie IBM MQ for z/OS, czy zezwolić na zakończenie połączenia. Ta opcja ustawia również sygnał wskazujący, że użytkownik chce być powiadamiany o nadejściu komunikatu.

- Określa, czy wywołanie nie powiedzie się, jeśli menedżer kolejek jest w stanie wygaszania.
- z/OS W systemie IBM MQ for z/OS, bez względu na to, czy wywołanie nie powiedzie się, jeśli połączenie jest w stanie wygaszania
- Określa, czy wymagana jest konwersja danych komunikatu aplikacji (patrz “Konwersja danych aplikacji” na stronie 893)
- Kolejność, w jakiej komunikaty i segmenty są pobierane z kolejki z/OS (z wyjątkiem IBM MQ for z/OS)
- Niezależnie od tego, czy są kompletne, komunikaty logiczne są dostępne tylko dla programu z/OS (z wyjątkiem produktu IBM MQ for z/OS)
- Określa, czy komunikaty w grupie mogą być pobierane tylko w przypadku, gdy dostępne są *wszystkie* komunikaty w grupie.
- Określenie, czy segmenty w komunikacie logicznym mogą być pobierane tylko w przypadku, gdy *wszystkie* segmenty w komunikacie logicznym są dostępne z/OS (z wyjątkiem IBM MQ for z/OS)

Jeśli pole *Options* zostanie pozostaw ustawione na wartość domyślną (MQGMO_NO_WAIT), wywołanie MQGET będzie działać w ten sposób:

- Jeśli nie ma komunikatu zgodnego z kryteriami wyboru w kolejce, wywołanie nie czeka na nadejście komunikatu, ale zostanie zakończone natychmiast. z/OS Również w produkcie IBM MQ for z/OSwywołanie nie ustawia sygnału żądającego powiadomienia, gdy pojawi się taki komunikat.
- Sposób, w jaki połączenie działa z punktami synchronizacji jest określany przez platformę:

Platforma	**Pod kontrolą punktu synchronizacji**
IBM i	Nie
Systemy UNIX 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 pominięcie wycofania.
- Wybrany komunikat jest usuwany z kolejki (nie jest przeglądane).
- 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 dotarcie komunikatu do kolejki w przypadku użycia opcji MQGMO_WAIT. Jeśli w czasie określonym w *WaitInterval* nie pojawi się żaden komunikat, wywołanie zakończy się i zwróci kod przyczyny wskazujący, że nie było komunikatu zgodnego z kryteriami wyboru w kolejce.

z/OS W systemie IBM MQ for z/OS, jeśli używana jest opcja MQGMO_SET_SIGNAL, pole *WaitInterval* określa czas, dla którego sygnał jest ustawiany.

Więcej informacji na temat tych opcji można znaleźć w sekcji “Oczekiwanie na komunikaty” na stronie 889 z/OS i “sygnalizowanie” na stronie 890 .

Signal1

Produkt Signal1 jest obsługiwany tylko w systemie z/OS IBM MQ for z/OS.

Jeśli opcja MQGMO_SET_SIGNAL jest używana do żądania powiadomienia o aplikacji po nadejściu odpowiedniego komunikatu, należy określić typ sygnału w polu *Signal1* (Typ sygnału). W produkcie IBM MQ na wszystkich innych platformach pole *Signal1* jest zarezerwowane, a jego wartość nie jest znacząca.

z/OS Więcej informacji na ten temat zawiera "sygnalizowanie" na stronie 890.

Signal2
Pole *Signal2* jest zarezerwowane na wszystkich platformach, a jego wartość nie jest znacząca.

z/OS Więcej informacji na ten temat zawiera sekcja "sygnalizowanie" na stronie 890.

ResolvedQName
ResolvedQName to pole wyjściowe, w którym menedżer kolejek zwraca nazwę kolejki (po rozstrzygnięcie dowolnego aliasu), z której został pobrany komunikat.

MatchOptions
Produkt *MatchOptions* steruje kryteriami wyboru dla komendy MQGET.

GroupStatus
GroupStatus wskazuje, czy pobrany komunikat znajduje się w grupie.

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 do komunikatu, który ma zostać zapełniony właściwościami komunikatu pobieranego z kolejki. Uchwyt został wcześniej utworzony za pomocą wywołania 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. Decydujesz, jak duże powinno być to na trzy sposoby:

1. Użytkownik może już wiedzieć, jaka długość komunikatów ma być oczekiwana w tym programie. Jeśli tak, należy określić bufor o tej wielkości.

   Można jednak użyć opcji MQGMO_ACCEPT_TRUNCATED_MSG w strukturze MQGMO, jeśli wywołanie MQGET ma być zakończone nawet wtedy, gdy komunikat jest zbyt duży dla buforu. W tym przypadku:

   - Bufor jest wypełniany tak dużą, jak i może być wstrzymana
   - Wywołanie zwraca kod zakończenia ostrzeżenia
   - Komunikat jest usuwany z kolejki (odrzucaj pozostałą część komunikatu) lub kursor przeglądania jest zaawansowany (jeśli przeglądasz kolejkę)
   - Rzeczywista długość komunikatu jest zwracana w produkcie *DataLength* .

   Bez tej opcji wywołanie nadal kończy się ostrzeżeniem, ale nie usuwa komunikatu z kolejki (lub wyprzedzają kursor przeglądania).
2. Oszacuj wielkość buforu (lub nawet podaj wielkość zero bajtów), a *nie* należy użyć opcji MQGMO_ACCEPT_TRUNCATED_MSG. Jeśli wywołanie MQGET nie powiedzie się (na przykład, ponieważ bufor jest za mały), długość komunikatu jest zwracana w parametrze **DataLength**

wywołania. (Bufor jest nadal zapełniany tak, jak większość komunikatu może być wstrzymana, ale przetwarzanie wywołania nie zostało zakończone.) Zapisz *MsgId* tego komunikatu, a następnie powtórz wywołanie MQGET, określając obszar buforu o prawidłowej wielkości, a *MsgId* , który został odnotowane od pierwszego wywołania.

Jeśli program obsługuje kolejkę, która jest również obsługiwana przez inne programy, jeden z tych innych programów może usunąć komunikat, który ma zostać usunięty, zanim program będzie mógł wydać kolejne wywołanie MQGET. Program może tracić czas na wyszukiwanie wiadomości, która już nie istnieje. Aby tego uniknąć, należy najpierw przeglądać kolejkę aż do znalezienia komunikatu, określając wartość *BufferLength* równą zero i używając opcji MQGMO_ACCEPT_TRUNCATED_MSG. Spowoduje to umieszczenie kursora w komunikacie, który ma zostać wyświetlony. Następnie można pobrać komunikat, wywołując komendę MQGET ponownie, podając opcję MQGMO_MSG_UNDER_CURSOR. Jeśli inny program usunie komunikat między wywołaniami przeglądania i usuwania, druga operacja MQGET nie powiedzie się natychmiast (bez przeszukiwania całej kolejki), ponieważ pod kursorem przeglądania nie ma żadnego komunikatu.

3. Atrybut *MaxMsgLength kolejka* określa maksymalną długość komunikatów akceptowanych dla tej kolejki. Atrybut *MaxMsgLength menedżer kolejek* określa maksymalną długość komunikatów akceptowanych dla tego menedżera kolejek. Jeśli nie wiadomo, jaka długość komunikatu jest oczekiwana, można uzyskać informacje na temat atrybutu **MaxMsgLength** (za pomocą wywołania MQINQ), a następnie określić bufor o tej wielkości.

   Aby uniknąć zmniejszonej wydajności, należy podjąć próbę jak najbliżej rzeczywistej wielkości buforu.

   Więcej informacji na temat atrybutu **MaxMsgLength** zawiera sekcja "Zwiększanie maksymalnej długości komunikatu" na stronie 883.

Kolejność, w jakiej komunikaty są pobierane z kolejki

Użytkownik może sterować kolejką, w której pobierane są komunikaty z kolejki. Ta sekcja zawiera informacje na temat opcji.

Priorytet

Program może przypisać priorytet do komunikatu, gdy umieszcza komunikat w kolejce (patrz "Priorytety komunikatów" na stronie 25). Komunikaty o równym priorytecie są przechowywane w kolejce w celu ich przybycia, a nie kolejności, w jakiej są zatwierdzane.

Menedżer kolejek przechowuje kolejki w ścisłej sekwencji FIFO (najpierw w pierwszej kolejności) lub w 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 otrzymać pierwszy komunikat z kolejki lub mogą otrzymać określony komunikat z kolejki, ignorując priorytet tych komunikatów. Program może na przykład chcieć przetworzyć odpowiedź na konkretny komunikat, który został przesłany wcześniej. Więcej informacji na ten temat zawiera sekcja "Uzyskiwanie określonego komunikatu" na stronie 877.

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 zostały umieszczone na zewnątrz jednostki pracy.
- Kolejka jest lokalna względem umieszczonej aplikacji

Jeśli warunki te nie są spełnione, a aplikacje są zależne od komunikatów pobieranych w określonej kolejności, aplikacje muszą zawierać informacje o kolejności w danych komunikatu lub 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 882.

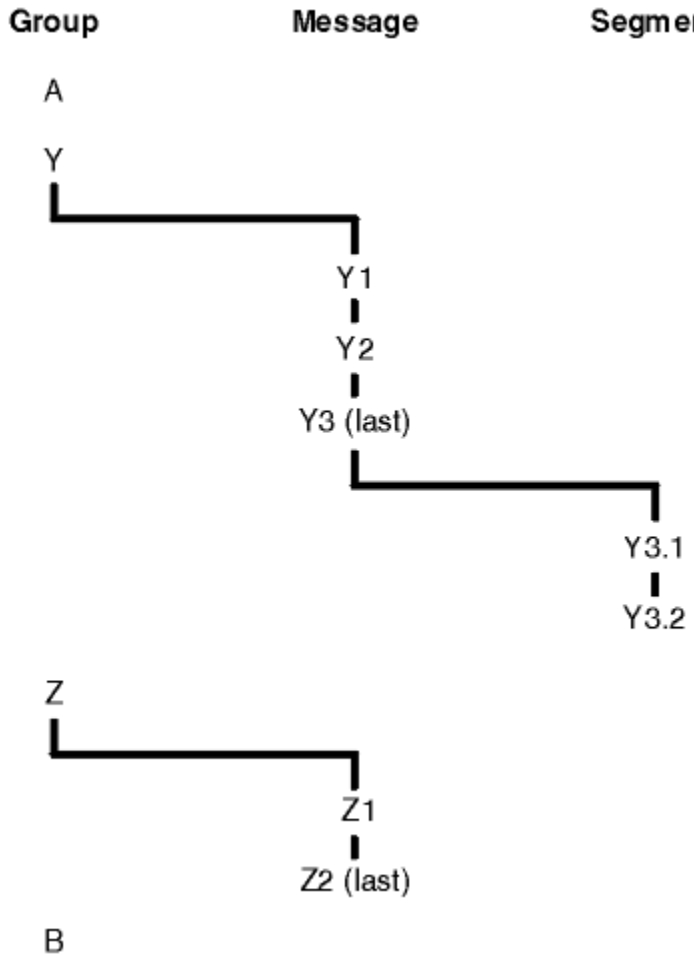
Uporządkowanie logiczne i fizyczne
Komunikaty w kolejkach mogą występować (w obrębie każdego poziomu priorytetu) w kolejności *fizycznej* lub *logicznej* .

Porządek fizyczny jest to kolejność, w jakiej komunikaty docierają do kolejki. Kolejność logiczna polega na tym, że wszystkie komunikaty i segmenty w grupie znajdują się w ich sekwencji logicznej, obok siebie, w pozycji określonej przez fizyczne położenie pierwszej pozycji należącej do grupy.

Opis grup, komunikatów i segmentów zawiera sekcja “Grupy komunikatów” na stronie 44. Te zlecenia fizyczne i logiczne mogą się różnić, ponieważ:

- Grupy mogą przybyć do miejsca docelowego w podobnych sytuacjach z różnych aplikacji, a tym samym utracić dowolny odrębny porządek fizyczny.
- Nawet w obrębie jednej grupy komunikaty mogą być wysyłane poza kolejną kolejką, ponieważ są one przekierowujące lub opóźnione w przypadku niektórych komunikatów w grupie.

Na przykład kolejność logiczna może wyglądać na rysunku Rysunek 66 na stronie 866:



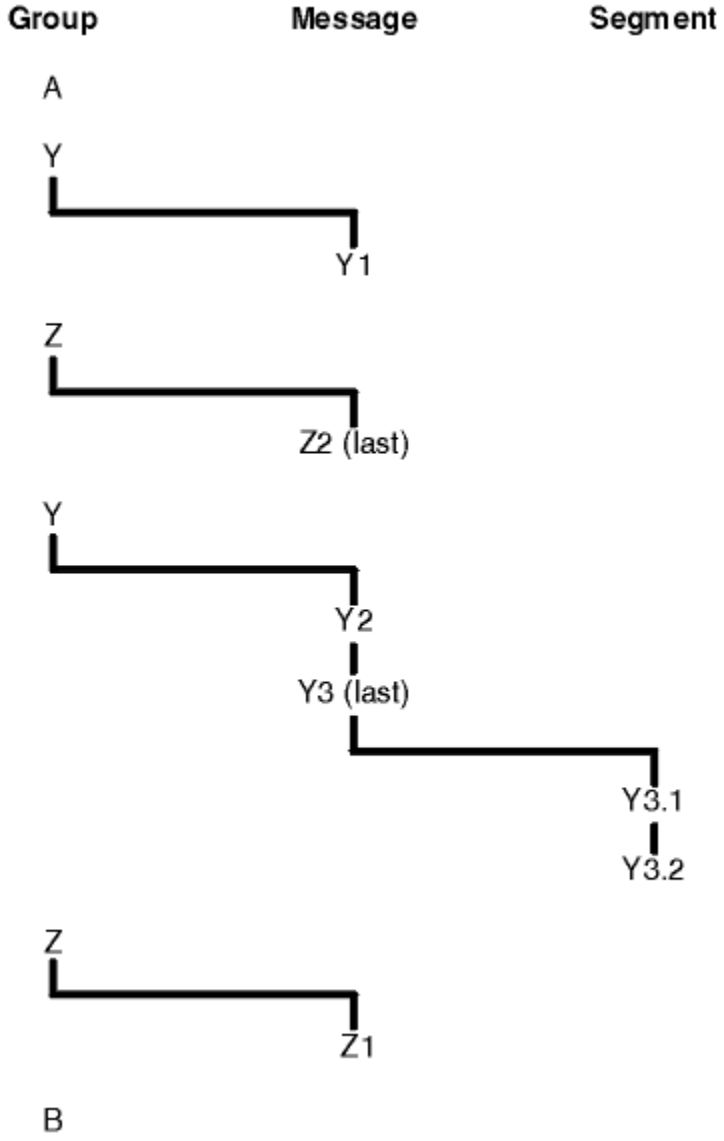
Rysunek 66. Porządek logiczny w kolejce

Komunikaty te mogą wystąpić w następującej kolejności logicznej w kolejce:

1. Komunikat A (nie w grupie)
2. Komunikat logiczny 1 grupy Y
3. Komunikat logiczny 2 grupy Y
4. Segment 1 of (ostatni) komunikat logiczny 3 grupy Y
5. (Ostatni) segment 2 z (ostatniego) komunikatu logicznego 3 grupy Y
6. Komunikat logiczny 1 grupy Z

7. (Ostatni) komunikat logiczny 2 grupy Z
8. Komunikat B (nie znajduje się w grupie)

Porządek fizyczny może być jednak zupełnie inny. Pozycja fizyczna elementu *first* w każdej grupie określa pozycję logiczną całej grupy. Na przykład, jeśli grupy Y i Z dotarły do podobnych czasów, a komunikat 2 grupy Z wyprzedzał komunikat 1 z tej samej grupy, porządek fizyczny będzie wyglądał podobnie do rysunku Rysunek 67 na stronie 867:



Rysunek 67. Porządek fizyczny w kolejce

Komunikaty te pojawiają się 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 grupy Y
5. Segment 1 of (ostatni) komunikat logiczny 3 grupy Y
6. (Ostatni) segment 2 z (ostatniego) komunikatu logicznego 3 grupy Y
7. Komunikat logiczny 1 grupy Z
8. Komunikat B (nie znajduje się w grupie)

Uwaga: W systemie IBM MQ for z/OSfizyczna kolejność komunikatów w kolejce nie jest gwarantowana, jeśli kolejka jest indeksowana przez GROUPID.

Podczas pobierania komunikatów można określić MQGMO_LOGICAL_ORDER, aby pobierać komunikaty w porządku logicznym, a nie w porządku fizycznym.

Jeśli wywoła wywołanie 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 MQGET z MQGMO_BROWSE_FIRST nie określa MQGMO_LOGICAL_ORDER, nie należy podawać następujących operacji MQGETs z MQGMO_BROWSE_NEXT.

Informacje o grupach i segmentach, które menedżer kolejek zachowuje dla wywołań MQGET, które przeglądną komunikaty w kolejce, są oddzielone od informacji o grupach i segmentach, które menedżer kolejek zachowuje dla wywołań MQGET, które usuwają komunikaty z kolejki. Jeśli zostanie określona wartość MQGMO_BROWSE_FIRST, menedżer kolejek ignoruje informacje o grupie i segmencie w celu przeglądania, a następnie skanuje kolejkę tak, jakby nie było żadnej bieżącej grupy i nie ma bieżącego komunikatu logicznego.

Uwaga: Nie należy używać wywołania MQGET w celu przeglądania *poza końcem* grupy komunikatów (lub komunikatu logicznego nie w grupie) bez określania parametru MQGMO_LOGICAL_ORDER. Na przykład, jeśli ostatni komunikat w grupie *poprzedza* pierwszy komunikat w grupie w kolejce, za pomocą komendy MQGMO_BROWSE_NEXT w celu przejrzenia poza koniec grupy, podanie wartości MQMO_MATCH_MSG_SEQ_NUMBER z parametrem *MsgSeqNumber* ustawionym na 1 (aby znaleźć pierwszy komunikat następnej grupy) ponownie zwróci pierwszy komunikat w grupie, który został już przeglądane. Może to nastąpić natychmiast lub kilka wywołań MQGET w późniejszym czasie (jeśli istnieją grupy, które się do tego zdarzają).

Aby uniknąć możliwości pętli nieskończonej, należy otworzyć kolejkę *dwukrotnie* w celu wyszukania:

- Użyj pierwszego uchwytu, aby przeglądać tylko pierwszy komunikat w każdej grupie.
- Użyj drugiego uchwytu, aby przeglądać tylko komunikaty należące do określonej grupy.
- Użyj opcji MQMO_ *, aby przenieść drugi kursor przeglądania na pozycję pierwszego kursora przeglądania, przed przeglądaniem komunikatów w grupie.
- Nie należy używać funkcji przeglądania MQGMO_BROWSE_NEXT poza końcem grupy.

Więcej informacji na ten temat można znaleźć w sekcji MQGET, MQMDi Reguły sprawdzania poprawności opcji MQI.

W przypadku większości aplikacji można wybrać porządkowanie logiczne lub fizyczne podczas przeglądania. Jeśli jednak użytkownik chce przełączać się między tymi trybami, należy pamiętać, że podczas pierwszego wydania przeglądania za pomocą komendy MQGMO_LOGICAL_ORDER, położenie w sekwencji logicznej jest ustanawiane.

Jeśli pierwszy element w grupie nie jest obecny w tym czasie, grupa, w której użytkownik nie jest w stanie, nie jest uznawana za część sekwencji logicznej.

Gdy kursor przeglądania znajduje się w grupie, może on być kontynuowany w obrębie tej samej grupy, nawet jeśli pierwszy komunikat zostanie usunięty. Początkowo nigdy nie można przenieść do grupy za pomocą komendy MQGMO_LOGICAL_ORDER, w której pierwszy element nie jest obecny.

MQPMO_LOGICAL_ORDER

Opcja MQPMO informuje menedżera kolejek o tym, w jaki sposób aplikacja umieszcza komunikaty w grupach i segmentach komunikatów logicznych. Można ją określić tylko w wywołaniu MQPUT. Nie jest ona poprawna w wywołaniu metody 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 opowiedziała menedżerowi kolejek, w jaki sposób umieszcza komunikaty w grupach i segmentach komunikatów logicznych, aplikacja nie musi obsługiwać i aktualizować informacji o grupach i segmentach na temat poszczególnych wywołań MQPUT, ponieważ menedżer kolejek przechowuje 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 ustawić pole *MsgFlags* tylko w deskryptywie MQMD, aby wskazać, kiedy komunikaty należą do grup lub są segmentami komunikatów logicznych, a także aby wskazać ostatni komunikat w grupie lub ostatnim segmencie komunikatu logicznego.

Po uruchomieniu grupy komunikatów lub komunikatu logicznego kolejne wywołania MQPUT muszą określać odpowiednie opcje MQMF_ * w produkcie *MsgFlags* w strukturze MQMD. Jeśli aplikacja podejmie próbę umieszczenia komunikatu, który nie znajduje się w grupie, gdy istnieje niezakończona grupa komunikatów lub komunikat, który nie jest segmentem, jeśli istnieje niezakończony komunikat logiczny, wywołanie kończy się niepowodzeniem z kodem przyczyny MQRC_INCOMPLETE_GROUP lub MQRC_INCOMPLETE_MSG, w zależności od przypadku. Menedżer kolejek zachowuje jednak informacje na temat bieżącej grupy komunikatów lub bieżącego komunikatu logicznego, a aplikacja może je zakończyć, wysyłając komunikat (prawdopodobnie bez danych komunikatu aplikacji), określając odpowiednio MQMF_LAST_MSG_IN_GROUP lub MQMF_LAST_SEGMENT, przed ponownym wywołaniem wywołania MQPUT w celu umieszczenia komunikatu, który nie znajduje się w grupie, albo nie jest to segment.

Rysunek 67 na stronie 867 przedstawia kombinacje opcji i flag, które są poprawne, oraz wartości pól *GroupId*, *MsgSeqNumber*i *Offset* używanych przez menedżer kolejek w każdym przypadku. Kombinacje opcji i opcji, które nie są wyświetlane w tabeli, są niepoprawne. Kolumny w tabeli mają następujące znaczenia: albo Tak, albo Nie:

PROTOKÓŁ ORD
Określa, czy opcja MQPMO_LOGICAL_ORDER jest określona w wywołaniu.

MIG
Określa, czy w wywołaniu jest określona opcja MQMF_MSG_IN_GROUP lub MQMF_LAST_MSG_IN_GROUP.

SEG
Określa, czy w wywołaniu jest określona opcja MQMF_SEGMENT lub MQMF_LAST_SEGMENT.

SEG OK
Określa, czy opcja MQMF_SEGMENTATION_ALLOWED jest określona w wywołaniu.

Grp
Określa, czy bieżąca grupa komunikatów istnieje przed wywołaniem.

Komunikat dziennika Cur
Określa, czy bieżący komunikat logiczny istnieje przed wywołaniem.

Pozostałe kolumny
Pokaż wartości używane przez menedżer kolejek. Poprzednio oznaczono wartość użytą dla pola w poprzednim komunikacie dla uchwytu kolejki.

Tabela 120. Opcje MQPUT odnoszące się do komunikatów w grupach i segmentach komunikatów logicznych

Opcje określone przez użytkownika	**Opcje określone przez użytkownika**	**Opcje określone przez użytkownika**	**Opcje określone przez użytkownika**	**Status grupy i dziennika-komunikat przed wywołaniem**	**Status grupy i dziennika-komunikat przed wywołaniem**	**Wartości, których używa menedżer kolejek**	**Wartości, których używa menedżer kolejek**	**Wartości, których używa menedżer kolejek**
PROTOKÓŁ ORD	MIG	SEG	SEG OK	Grp	Komunikat dziennika Cur	**GroupI**d	**MsgSeqNumber**	**Offset**
Tak	Nie	Nie	Nie	Nie	Nie	MQGI_NONE	1	0
Tak	Nie	Nie	Tak	Nie	Nie	Identyfikator nowej grupy	1	0
Tak	Nie	Tak	Albo	Nie	Nie	Identyfikator nowej grupy	1	0
Tak	Nie	Tak	Albo	Nie	Tak	Poprzedni identyfikator grupy	1	Poprzednie przesunięcie + długość poprzedniego segmentu
Tak	Tak	Albo	Albo	Nie	Nie	Identyfikator nowej 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 + długość poprzedniego segmentu
Nie	Nie	Nie	Nie	Albo	Albo	MQGI_NONE	1	0
Nie	Nie	Nie	Tak	Albo	Albo	Identyfikator nowej grupy, jeśli wartość MQGI_NONE, wartość else w polu	1	0
Nie	Nie	Tak	Albo	Albo	Albo	Identyfikator nowej grupy, jeśli wartość MQGI_NONE, wartość else w polu	1	Wartość w polu

Tabela 120. Opcje MQPUT odnoszące się do komunikatów w grupach i segmentach komunikatów logicznych (kontynuacja)

Opcje określone przez użytkownika	Opcje określone przez użytkownika	Opcje określone przez użytkownika	Opcje określone przez użytkownika	Status grupy i dziennika-komunikat przed wywołaniem	Status grupy i dziennika-komunikat przed wywołaniem	Wartości, których używa menedżer kolejek	Wartości, których używa menedżer kolejek	Wartości, których używa menedżer kolejek
Nie	Tak	Nie	Albo	Albo	Albo	Identyfikator nowej grupy, jeśli wartość MQGI_NONE, wartość else w polu	Wartość w polu	0
Nie	Tak	Tak	Albo	Albo	Albo	Identyfikator nowej grupy, jeśli wartość MQGI_NONE, wartość else w polu	Wartość w polu	Wartość w polu

Uwaga:

- Parametr MQPMO_LOGICAL_ORDER nie jest poprawny w wywołaniu MQPUT1 .
- W przypadku pola *MsgId* menedżer kolejek generuje nowy identyfikator komunikatu, jeśli określono wartość MQPMO_NEW_MSG_ID lub MQMI_NONE, a w przeciwnym razie używa wartości w polu.
- W przypadku pola *CorrelId* menedżer kolejek generuje nowy identyfikator korelacji, jeśli określono wartość MQPMO_NEW_CORREL_ID, a w przeciwnym razie używa wartości w polu.

Jeśli określono parametr MQPMO_LOGICAL_ORDER, menedżer kolejek wymaga, aby wszystkie komunikaty w grupie i segmentach w komunikacie logicznym były umieszczane z taką samą wartością w polu *Persistence* w strukturze MQMD, co oznacza, że wszystkie muszą być trwałe lub wszystkie muszą być nietrwałe. Jeśli ten warunek nie jest spełniony, wywołanie MQPUT nie powiedzie się z kodem przyczyny MQRC_INCONSISTENT_PERSISTENCE.

Opcja MQPMO_LOGICAL_ORDER wpływa na jednostki pracy w następujący sposób:

- Jeśli pierwszy komunikat fizyczny w grupie lub komunikat logiczny jest umieszczany w jednostce pracy, wszystkie pozostałe komunikaty fizyczne w grupie lub komunikacie logicznym muszą być umieszczone w jednostce pracy, o ile ten sam uchwyt kolejki jest używany. Nie muszą one jednak być umieszczane w tej samej jednostce pracy, co pozwala na dzielenie grupy komunikatów lub komunikatów logicznych składającej 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 jest umieszczany w jednostce pracy, żaden inny komunikat fizyczny w grupie lub komunikat logiczny nie może zostać umieszczony w jednostce pracy, jeśli ten sam uchwyt kolejki jest używany.

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 pełnych grup komunikatów ani kompletnych komunikatów logicznych. Obowiązkiem aplikacji jest zapewnienie, że pola *GroupId*, *MsgSeqNumber*, *Offset*i *MsgFlags* mają odpowiednie wartości.

Za pomocą tej techniki można restartować grupę komunikatów lub komunikat logiczny w środku, po wystąpieniu awarii systemu. Po zrestartowaniu systemu aplikacja może ustawić pola *GroupId*, *MsgSeqNumber*, *Offset*, *MsgFlags*i *Persistence* na odpowiednie wartości, a następnie wywołać wywołanie MQPUT z zestawem MQPMO_SYNCPOINT lub MQPMO_NO_SYNCPOINT zgodnie z wymaganiami, ale bez określania parametru MQPMO_LOGICAL_ORDER. Jeśli to wywołanie powiedzie się, menedżer kolejek zachowuje informacje o grupie i segmencie, a kolejne wywołania MQPUT używające tego uchwytu kolejki mogą określać parametr MQPMO_LOGICAL_ORDER jako normalny.

Informacje o grupach i segmentach, które menedżer kolejek zachowuje dla wywołania MQPUT, są oddzielone od informacji o grupach i segmentach, które są zachowane dla wywołania MQGET.

W przypadku dowolnego uchwytu kolejki aplikacja może łączyć wywołania MQPUT, które określają parametr MQPMO_LOGICAL_ORDER z wywołaniami MQPUT, które nie są, ale należy zwrócić uwagę na następujące punkty:

- Jeśli parametr MQPMO_LOGICAL_ORDER nie zostanie określony, każde pomyślne wywołanie MQPUT powoduje, że menedżer kolejek ustawia informacje o grupach i segmentach dla uchwytu kolejki na wartości określone przez aplikację, zastępując istniejące informacje o grupach i segmentach zachowane przez menedżer kolejek dla uchwytu kolejki.
- Jeśli parametr MQPMO_LOGICAL_ORDER nie zostanie określony, wywołanie nie powiedzie się, jeśli istnieje bieżąca grupa komunikatów lub komunikat logiczny. Wywołanie może zakończyć się powodzeniem z kodem zakończenia MQCC_WARNING. Tabela 121 na stronie 872 przedstawia różne przypadki, które mogą wystąpić. W takich przypadkach, jeśli kod zakończenia nie jest kodem MQCC_OK, kod przyczyny jest jednym z następujących (odpowiednio):
  - 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 121. Wynik, gdy wywołanie MQPUT lub MQCLOSE nie jest spójne z informacjami o grupach i segmentach

Bieżące połączenie to	Poprzednie wywołanie było MQPUT z MQPMO_LOGICAL_ORDER	Poprzednie wywołanie było MQPUT bez MQPMO_LOGICAL_ORDER
MQPUT z MQPMO_LOGICAL_ORDER	MQCC_FAILED	MQCC_FAILED
MQPUT bez MQPMO_LOGICAL_ORDER	MQCC_WARNING,	MQCC_OK
MQCLOSE z niezakończonym komunikatem grupowym lub logicznym	MQCC_WARNING,	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 aplikację z potrzeby zarządzania informacjami o grupach i segmentach, ponieważ menedżer kolejek zarządza tą informacją. Jednak wyspecjalizowane aplikacje mogą wymagać większej kontroli niż określona przez opcję MQPMO_LOGICAL_ORDER, która może zostać osiągnięta przez niepodanie tej opcji. Jeśli tak jest, należy upewnić się, że pola *GroupId*, *MsgSeqNumber*, *Offset*i *MsgFlags* w strukturze MQMD są ustawione poprawnie, przed każdą wywołaniem MQPUT lub MQPUT1 .

Na przykład aplikacja, która chce przekazywać komunikaty fizyczne, które otrzymuje, bez względu na to, czy te komunikaty znajdują się w grupach, czy w segmentach komunikatów logicznych, nie może określać parametru MQPMO_LOGICAL_ORDER z dwóch powodów:

- Jeśli komunikaty są pobierane i umieszczane w porządku, podanie wartości MQPMO_LOGICAL_ORDER powoduje przypisanie do komunikatów nowego identyfikatora grupy, co może utrudnić lub uniemożliwić inicjatorowi komunikaty korelowanie wszystkich komunikatów odpowiedzi lub raportów, które wynikają z grupy komunikatów.
- W złożonej sieci z wieloma ścieżkami między wysyłającym i odbierającym menedżery kolejek komunikaty fizyczne mogą być odbierane poza kolejnością. Nie podając wartości MQPMO_LOGICAL_ORDER i MQGMO_LOGICAL_ORDER w wywołaniu MQGET, aplikacja przekazujący może pobrać i przesłać każdy komunikat fizyczny zaraz po jego nadejściu, bez oczekiwania na nadejście kolejnego w kolejności logicznej.

Aplikacje, które generują komunikaty raportów w przypadku 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 przy użyciu dowolnej z pozostałych opcji MQPMO_ *.

Umieszczanie grup uporządkowanych logicznie w kolejce klastrowej (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. Ma to miejsce, w którym ruch aplikacji nie jest równoważeniem obciążenia w wielu instancjach kolejki klastra. Aby włączyć równoważenie obciążenia przy jednoczesnym zachowaniu grup komunikatów w stanie nienaruszonym, należy ustawić następujące opcje:

- Wywołanie MQPUT musi określać MQPMO_LOGICAL_ORDER.
- Wywołanie MQOPEN musi określać jedną z dwóch następujących opcji:
  - MQOO_BIND_ON_GROUP
  - MQOO_BIND_AS_Q_DEF, a definicja kolejki musi określać DEFBIND (GROUP)

Równoważenie obciążenia jest następnie sterowane *między grupami* komunikatów bez konieczności użycia komendy MQCLOSE i MQOPEN w kolejce. *Między grupami* oznacza, że parametr MQMF_MSG_IN_GROUP jest ustawiony w strukturze MQMD (v2) lub MQMDE, a w toku nie istnieje częściowa kompletna grupa. Gdy grupa jest w toku, rozstrzygnięty menedżer kolejek i nazwa kolejki w uchwycie obiektu są ponownie wykorzystywane.

Jeśli poprzedni komunikat miał ustawioną wartość MQPMO_LOGICAL_ORDER i/lub MQMF_MSG_IN_GROUP, ale bieżący komunikat nie jest częścią grupy, wywołanie PUT kończy się niepowodzeniem z parametrem MQRC_INCOMPLETE_GROUP.

Jeśli pojedyncza operacja MQPUT nie określa parametru MQPMO_LOGICAL_ORDER, a żadna bieżąca grupa nie jest aktywna, to równoważenie obciążenia jest kierowane dla tego komunikatu (tak jakby wywołanie MQOPEN określone zostało określone w tabeli MQOO_BIND_NOT_FIXED).

Nie ma miejsca do ponownego przydzielenia komunikatów powiązanych z miejscem docelowym za pomocą komendy MQOO_BIND_ON_GROUP. Więcej informacji na temat ponownego przydzielania można znaleźć w sekcji “Grupy komunikatów” na stronie 44.

Grupowanie komunikatów logicznych

Istnieją dwie główne przyczyny korzystania z komunikatów logicznych w grupie:

- Może być konieczne przetworzenie komunikatów w określonej kolejności.
- Może być konieczne przetworzenie każdego komunikatu w grupie w pokrewny sposób.

W każdym z tych przypadków należy pobrać całą grupę z tą samą instancją aplikacji pobierając.

Załóżmy na przykład, że grupa składa się z czterech komunikatów logicznych. Umieszczenie aplikacji wygląda następująco:

```
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 pobierający określa opcję MQGMO_ALL_MSGS_AVAILABLE dla pierwszego komunikatu w grupie. Zapewnia to, że przetwarzanie nie zostanie uruchomione, dopóki nie zostaną odebrane wszystkie komunikaty w grupie. Opcja MQGMO_ALL_MSGS_AVAILABLE jest ignorowana w przypadku kolejnych komunikatów w grupie.

Po pobraniu pierwszego komunikatu logicznego grupy można użyć komendy MQGMO_LOGICAL_ORDER, aby upewnić się, że pozostałe komunikaty logiczne grupy są pobierane w kolejności.

Tak więc, pobieranie aplikacji 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
```

Więcej informacji na temat grupowania komunikatów można znaleźć w sekcji “Segmentacja aplikacji komunikatów logicznych” na stronie 886 i “Umieszczanie i pobieranie grupy, która obejmuje jednostki pracy” na stronie 874.

Więcej informacji na temat zezwalania aplikacji na żądanie, aby grupa komunikatów była przydzielona do tej samej instancji docelowej dla kolejek klastra, należy zapoznać się z informacjami w sekcji DefBind.

Umieszczanie i pobieranie grupy, która obejmuje jednostki pracy

W poprzednim przypadku komunikaty lub segmenty nie mogą zostać uruchomione w celu opuszczenia węzła (jeśli jego miejsce docelowe jest zdalne) lub uruchomienie do pobrania do momentu umieszczenia całej grupy i zatwierdzenia jednostki pracy. Może to nie być to, czego chcesz, jeśli zajmuje dużo czasu, aby umieścić całą grupę, lub jeśli obszar kolejki jest ograniczony w węźle. Aby przezwyciężyć to, należy umieścić grupę w kilku jednostkach pracy.

Jeśli grupa jest umieszczana w wielu jednostkach pracy, niektóre z grup mogą być zatwierdzane nawet wtedy, gdy wprowadzanie aplikacji nie powiedzie się. W związku z tym aplikacja musi zapisywać informacje o statusie, zatwierdzane przy użyciu każdej jednostki pracy, którą może użyć po restarcie w celu wznowienia niekompletnej grupy. Najprostsze miejsce do zarejestrowania tych informacji znajduje się w kolejce STATUS. Jeśli kompletna grupa została pomyślnie wstawiona, kolejka STATUS jest pusta.

Jeśli segmentacja jest zaangażowana, logika jest podobna. W takim przypadku **StatusInfo** musi zawierać *Offset*.

Poniżej przedstawiono 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 ułożona, a kolejka STATUS jest pusta. Jeśli nie, grupa musi zostać wznowiona w punkcie wskazanym przez informacje o statusie. MQPMO_LOGICAL_ORDER nie może być użyty do pierwszego umieszczenia, ale może później.

Ponowne uruchomienie przetwarzania 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 pobierających można rozpocząć przetwarzanie komunikatów w grupie, zanim cała grupa zostanie nadesłana. Powoduje to zwiększenie czasów odpowiedzi w komunikatach w grupie, a także oznacza, że pamięć nie jest wymagana dla całej grupy. W celu zrealizowania korzyści należy użyć kilku jednostek pracy dla każdej grupy komunikatów. Ze względów odzyskiwania konieczne jest pobranie każdego komunikatu w ramach jednostki pracy.

Podobnie jak w przypadku aplikacji wprowadzania danych, wymagane jest, aby informacje o statusie były zapisywane w dowolnym miejscu automatycznie, ponieważ każda jednostka pracy jest zatwierdzana. Najprostsze miejsce rejestrowania tych informacji znajduje się w kolejce 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 dla kolejki statusu jest segmentem komunikatu (czyli przez ustawienie flagi MQMF_SEGMENT), zamiast umieszczania kompletnego nowego komunikatu dla każdej jednostki pracy. W ostatniej jednostce pracy segment końcowy jest umieszczany w kolejce statusu z parametrem MQMF_LAST_SEGMENT, a następnie informacje o statusie są kasowane za pomocą komendy MQGET, która określa parametr MQGMO_COMPLETE_MSG.

Podczas przetwarzania restartu zamiast pojedynczej operacji MQGET, aby uzyskać możliwy komunikat o statusie, należy przejrzeć kolejkę statusu za pomocą komendy MQGMO_LOGICAL_ORDER, aż do ostatniego segmentu (czyli do momentu, w którym nie zostaną zwrócone żadne kolejne segmenty).

W pierwszej jednostce pracy po restarcie należy również określić przesunięcie jawnie podczas umieszczania segmentu statusu.

W poniższym przykładzie rozważamy tylko komunikaty w grupie, zakładając, że bufor aplikacji jest zawsze na tyle duży, aby pomieścić cały komunikat, niezależnie od tego, czy komunikat został podzielony na segmenty. W związku z tym w każdej operacji MQGET określono parametr MQGMO_COMPLETE_MSG. Te same zasady mają zastosowanie w przypadku podziału segmentacji (w tym przypadku element StatusInfo musi zawierać *Offset*).

Dla prostoty zakładamy, że maksymalnie 4 wiadomości są pobierane w ramach jednej jednostki pracy:

```
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 wczytana, a kolejka STATUS jest pusta. Jeśli nie, grupa musi zostać wznowiona w punkcie wskazanym przez informacje o statusie. Nie można użyć komendy MQGMO_LOGICAL_ORDER dla pierwszego pobrania, ale może to być później.

Ponowne uruchomienie przetwarzania 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 określonego komunikatu

Istnieje wiele sposobów uzyskiwania określonego komunikatu z kolejki. Są to następujące elementy: wybór opcji MsgId i CorrelId, wybór opcji GroupId, MsgSeqNumber i Offset, a następnie wybranie opcji MsgToken. Po otwarciu kolejki można również użyć łańcucha wyboru.

Aby uzyskać konkretny komunikat z kolejki, należy użyć pól `MsgId` i `CorrelId` struktury MQMD. Jednak aplikacje mogą jawnie ustawić te pola, tak więc podane wartości mogą nie identyfikować unikalnego komunikatu. Tabela 122 na stronie 877 pokazuje, który komunikat jest pobierany dla możliwych ustawień tych pól. Te pola są ignorowane na wejściu, jeśli określono wartość MQGMO_MSG_UNDER_CURSOR w parametrze **GetMsgOpts** wywołania MQGET.

Tabela 122. Korzystanie z identyfikatorów komunikatów i korelacji

Aby pobrać ...	**MsgId**	**CorrelId**
Pierwszy komunikat w kolejce	MQMI_NONE	MQCI_NONE
Pierwszy komunikat zgodny z *MsgId*	Niezerowe	MQCI_NONE
Pierwszy komunikat zgodny z *CorrelId*	MQMI_NONE	Niezerowe
Pierwszy komunikat zgodny z *MsgId* i *CorrelId*	Niezerowe	Niezerowe

W każdym przypadku wartość *pierwsza* oznacza pierwszy komunikat, który spełnia kryteria wyboru (chyba że określono parametr MQGMO_BROWSE_NEXT, gdy oznacza to *następny* komunikat w sekwencji spełniający kryteria wyboru).

Po powrocie wywołanie MQGET ustawia pola *MsgId* i *CorrelId* na identyfikatory komunikatów i korelacji zwróconego komunikatu (jeśli istnieje).

Jeśli w polu *Version* struktury MQMD zostanie ustawiona wartość 2, można użyć pól *GroupId*, *MsgSeqNumber*i *Offset* . Tabela 123 na stronie 877 pokazuje, który komunikat jest pobierany dla możliwych ustawień tych pól.

Tabela 123. Korzystanie z identyfikatora grupy

Aby pobrać ...	**opcje dopasowywania**
Pierwszy komunikat w kolejce	MQMO_NONE
Pierwszy komunikat zgodny z `MsgId`	MQMO_MATCH_MSG_ID
Pierwszy komunikat zgodny z `CorrelId`	MQMO_MATCH_KORELID
Pierwszy komunikat zgodny z `GroupId`	MQMO_MATCH_GROUP_ID
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_MATCH_OFFSET

Uwagi:

1. MQMO_MATCH_XXX oznacza, że pole *XXX* w strukturze MQMD jest ustawione na wartość, która ma zostać dopasowana.
2. Opcje MQMO mogą być używane w połączeniu. Na przykład: MQMO_MATCH_GROUP_ID, MQMO_MATCH_MSG_SEQ_NUMBER i MQMO_MATCH_OFFSET mogą być używane razem w celu nadania segmentu identyfikowanego za pomocą pól `GroupId`, `MsgSeqNumber`i `Offset` .
3. Jeśli zostanie określona wartość MQGMO_LOGICAL_ORDER, wpłynie to na komunikat, który ma zostać pobrany, ponieważ opcja zależy od informacji o stanie kontrolowanej przez uchwyt kolejki. Więcej informacji na ten temat można znaleźć w sekcji "Uporządkowanie logiczne i fizyczne" na stronie 866 i Opcje.

Wywołanie MQGET zwykle pobiera pierwszy komunikat z kolejki. Jeśli podczas korzystania z wywołania MQGET zostanie określony konkretny komunikat, menedżer kolejek musi przeszukac kolejkę, dopóki nie znajdzie tego komunikatu. Może to mieć wpływ na wydajność aplikacji.

Jeśli używana jest wersja 2 lub nowsza struktury MQGMO i nie określono opcji MQMO_MATCH_MSG_ID lub MQMO_MATCH_CORREL_ID, nie ma potrzeby resetowania pól `MsgId` lub `CorrelId` między MQGET.

z/OS W systemie IBM MQ for z/OSatrybut kolejki `IndexType` może być używany w celu zwiększenia szybkości operacji MQGET w kolejce. Więcej informacji na ten tematzawiera sekcja "Typ indeksu" na stronie 882.

Konkretny komunikat można pobrać z kolejki, określając jego element MsgToken oraz element MatchOption MQMO_MATCH_MSG_TOKEN w strukturze MQGMO. Element MsgToken jest zwracany przez wywołanie MQPUT, które pierwotnie umieło umieścić ten komunikat w kolejce, lub przez poprzednie operacje MQGET i pozostaje stałe, chyba że menedżer kolejek zostanie zrestartowany.

Jeśli interesujesz się tylko podzbiorem komunikatów w kolejce, możesz określić, które komunikaty mają być przetwarzane przy użyciu łańcucha wyboru z wywołaniem MQOPEN lub MQSUB. Komenda MQGET pobiera następnie następny komunikat, który spełnia ten łańcuch wyboru. Więcej informacji na temat łańcuchów wyboru zawiera sekcja "Selektory" na stronie 30.

Zwiększanie wydajności nietrwałych komunikatów

Gdy klient wymaga komunikatu z serwera, wysyła żądanie do serwera. Wysyła on osobne żądanie dla każdego z komunikatów, które konsumuje. Aby zwiększyć wydajność klientów korzystających z nietrwałych komunikatów przez uniknięcie konieczności wysyłania tych komunikatów żądań, klient może być skonfigurowany do używania *odczytu z wyprzedzeniem*. Odczyt z wyprzedzeniem umożliwia wysyłanie komunikatów do klienta bez konieczności żądania ich żądania.

Gdy opcja odczytu z wyprzedzeniem jest włączona, komunikaty są wysyłane do buforu pamięci na kliencie o nazwie *bufor odczytu z wyprzedzeniem*. Klient będzie miał bufor odczytu z wyprzedzeniem dla każdej kolejki, która została otwarta z włączoną obsługą odczytu z wyprzedzeniem. Komunikaty w buforze odczytu z wyprzedzeniem nie są utrwalane. Klient okresowo aktualizuje serwer, podając informacje o ilości zużywanej przez niego danych.

W przypadku wywołania MQOPEN z opcją MQOO_READ_AHEAD klient IBM MQ umożliwia odczyt z wyprzedzeniem, jeśli spełnione są pewne warunki. Są one następujące:

- Klient i zdalny menedżer kolejek muszą być w wersji IBM WebSphere MQ 7.0 lub nowszej.
- 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 korzystania z nietrwałych komunikatów z aplikacji klienckiej. Ta poprawa wydajności jest dostępna zarówno dla aplikacji MQI, jak i aplikacji JMS . Aplikacje klienckie używające wywołania MQGET lub asynchronicznego będą korzystać z udoskonaleń 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 są wymagane w celu zachowania spójności między wywołaniami MQGET, gdy funkcja odczytu z wyprzedzeniem jest włączona. Jeśli klient zmienia kryteria wyboru między wywołaniami MQGET, komunikaty zapisywane w buforze odczytu z wyprzedzeniem pozostaną bez zmian w buforze odczytu z wyprzedzeniem klienta.

Jeśli zaległy dziennik komunikatów z poprzednimi kryteriami wyboru nie jest już wymagany, można ustawić konfigurowalny przedział czasu czyszczenia na kliencie, aby automatycznie wyczyścić te komunikaty od klienta. Przedział czasu czyszczenia jest jedną z grup opcji strojenia odczytu z wyprzedzeniem określonych przez klienta. Możliwe jest dostrojenie tych opcji w celu spełnienia wymagań.

Jeśli aplikacja kliencka zostanie zrestartowana, komunikaty w buforze odczytu z wyprzedzeniem mogą zostać utracone. I odwrotnie, komunikat, który został przeniesiony do buforu odczytu z wyprzedzeniem, mógł zostać usunięty z kolejki bazowej. Nie powoduje to usunięcia go z buforu, dlatego wywołanie MQGET z użyciem 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 odczytywawy przed klientem. Odczyt z wyprzedzeniem nie generuje informacji o rachunkach i statystykach, gdy jest on włączony.

Korzystanie z odczytu z wyprzedzeniem przy użyciu funkcji przesyłania komunikatów subskrypcji

Gdy aplikacja subskrybująca określa kolejkę docelową, do której wysyłane są publikacje, wartość DEFREADA określonej kolejki jest używana jako domyślna wartość odczytu z wyprzedzeniem.

Gdy aplikacja subskrybująca żąda, aby produkt IBM MQ zarządza miejscem docelowym, do którego są wysyłane publikacje, kolejka zarządzana jest tworzona jako kolejka dynamiczna w oparciu o predefiniowaną kolejkę modelową. Jest to wartość DEFREADA kolejki modelowej, która jest używana jako domyślna wartość odczytu z wyprzedzeniem. Domyślne kolejki modelowe SYSTEM.DURABLE.PUBLICATIONS.MODEL lub SYSTEM.NONDURABLE.PUBLICATIONS.MODEL jest używany, chyba że kolejka modelowa jest zdefiniowana dla tego lub nadrzędnego tematu.

Pojęcia pokrewne

“Strojenie wydajności dla nietrwałych komunikatów w systemie AIX” na stronie 881
Jeśli używany jest produkt AIX V5.3 lub nowszy, należy rozważyć ustawienie parametru strojenia w celu użycia pełnej wydajności dla nietrwałych komunikatów.

Zadania pokrewne

“Włączanie i wyłączanie odczytu z wyprzedzeniem” na stronie 881
Domyślnie funkcja odczytu z wyprzedzeniem jest wyłączona. Istnieje możliwość włączenia odczytu z wyprzedzeniem na poziomie kolejki lub aplikacji.

Odsyłacze pokrewne

“Opcje MQGET i odczyt z wyprzedzeniem” na stronie 879
Nie wszystkie opcje MQGET są obsługiwane, jeśli funkcja odczytu z wyprzedzeniem jest włączona. Niektóre opcje są wymagane w celu zachowania spójności między wywołaniami MQGET.

Opcje MQGET i odczyt z wyprzedzeniem
Nie wszystkie opcje MQGET są obsługiwane, jeśli funkcja odczytu z wyprzedzeniem jest włączona. Niektóre opcje są wymagane w celu zachowania spójności 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:

- Klient i zdalny menedżer kolejek muszą być w wersji IBM WebSphere MQ 7.0 lub nowszej.

- 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 tym, które opcje są obsługiwane w przypadku odczytu z wyprzedzeniem oraz czy można je zmieniać między wywołaniami MQGET.

Tabela 124. Opcje MQGET i odczyt z wyprzedzeniem

Wartości i opcje MQGET	**Dozwolone, gdy funkcja odczytu z wyprzedzeniem jest włączona i może być zmieniana między wywołaniami MQGET** [5]	**Dozwolone, gdy funkcja odczytu z wyprzedzeniem jest włączona, ale nie może być zmieniana między wywołaniami MQGET** [1]	**Opcje MQGET, które nie są dozwolone, gdy funkcja odczytu z wyprzedzeniem jest włączona** [2]
Wartości MQGET MQMD	MsgId [3] CorrelId [3]	Kodowanie CodedCharSetId	
Opcje MQGMO MQGET	• MQGMO_NO_WAIT • MQGMO_BROWSE_MESSAGE _UNDER_CURSOR • MQGMO_BROWSE_FIRST • MQGMO_BROWSE_NEXT • MQGMO_FAIL_IF_QUIESCING	• MQGMO_SYNCPOINT_IF _PERSISTENT • MQGMO_NO_SYNCPOINT • MQGMO_ACCEPT_OBCIĘTO _MSG • MQGMO_CONVERT	• MQGMO_SET_SIGNAL • MQGMO_SYNCPOINT, • MQGMO_MARK_SKIP _BACKOUT • MQGMO_MSG_UNDER _CURSOR [4] • Blokada MQGMO_LOCK • MQGMO_UNLOCK • MQGMO_LOGICAL_ORDER • MQGMO_COMPLETE_MSG • MQGMO_ALL_MSGS_AVAILABLE • MQGMO_ALL_SEGMENTS_ JEST 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 wartości parametrów MsgId i CorrelId aplikacji klienckiej między wywołaniami MQGET, komunikaty z poprzednimi wartościami mogły już zostać wysłane do klienta i pozostaną w buforze odczytu z wyprzedzeniem klienta do czasu zużytego (lub automatycznego wyczyszczenia).
4. Opcja MQGMO_MSG_UNDER_CURSOR nie jest dostępna, jeśli włączony jest odczyt z wyprzedzeniem. Funkcja odczytu z wyprzedzeniem jest wyłączona, gdy podczas otwierania kolejki określone są obie opcje MQOO_BROWSE i jedna z opcji MQOO_INPUT_SHARED lub MQOO_INPUT_EXCLUSIVE.
5. Jeśli opcja odczytu z wyprzedzeniem jest włączona, pierwsza komenda MQGET określa, czy komunikaty mają być przeglądane, czy też mają być wysyłane z kolejki. Jeśli aplikacja kliencka używa komendy MQGET ze zmienioną opcją, na przykład przy próbie wyszukania po początkowym dostawie lub próby pobrania za pomocą przeglądania początkowego, zwracany jest kod przyczyny MQRC_OPTIONS_CHANGED.

Jeśli klient zmienia kryteria wyboru między wywołaniami MQGET, komunikaty zapisywane w buforze odczytu z wyprzedzeniem, które są zgodne z początkowym kryterium wyboru, nie są wykorzystywane przez aplikację kliencką i pozostają bez zmian w buforze odczytu z wyprzedzeniem klienta. W sytuacjach, w których bufor odczytu z wyprzedzeniem klienta zawiera wiele komunikatów osieroconych, korzyści związane z odczytanie z wyprzedzeniem są tracone, a dla każdego zużytego komunikatu wymagane jest oddzielne żądanie do serwera. Aby określić, czy funkcja odczytu z wyprzedzeniem jest używana wydajnie, można użyć parametru statusu połączenia (READA).

Odczyt z wyprzedzeniem można zahamować, jeśli aplikacja jest żądana przez aplikację z powodu niezgodnych opcji określonych w pierwszym wywołaniu MQGET. W tej sytuacji status połączenia wskazuje odczyt z wyprzedzeniem jako zahamowany.

Jeśli z powodu tych ograniczeń dla wywołania MQGET użytkownik zdecyduje, że projekt aplikacji klienckiej nie nadaje się do odczytu z wyprzedzeniem, należy określić opcję MQOPEN MQOO_READ_AHEAD_NO. Alternatywnie ustaw domyślną wartość odczytu z wyprzedzeniem dla otwieranej kolejki, która została zmieniona na NO lub DISABLED.

Włączanie i wyłączanie odczytu z wyprzedzeniem
Domyślnie funkcja odczytu z wyprzedzeniem jest wyłączona. Istnieje możliwość włączenia odczytu z wyprzedzeniem 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:

- Klient i zdalny menedżer kolejek muszą być w wersji IBM WebSphere MQ 7.0 lub nowszej.
- 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, ustaw atrybut kolejki DEFREADA na YES.
- Aby skonfigurować odczyt z wyprzedzeniem na poziomie aplikacji:
  - w celu użycia odczytu z wyprzedzeniem wszędzie tam, gdzie to możliwe, należy użyć opcji MQOO_READ_AHEAD w wywołaniu funkcji MQOPEN. Nie jest możliwe, aby aplikacja kliencka używała odczytu z wyprzedzeniem, jeśli atrybut kolejki DEFREADA został ustawiony na wartość DISABLED.
  - w celu użycia odczytu z wyprzedzeniem tylko wtedy, gdy funkcja odczytu z wyprzedzeniem jest włączona 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, ustawiając atrybut kolejki, DEFREADA na NO, jeśli nie chcesz, aby odczyt z wyprzedzeniem był używany, jeśli nie jest wymagany przez aplikację kliencką, lub WYŁĄCZONY, jeśli nie chcesz, aby odczyt z wyprzedzeniem był 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 tego, co dzieje się z komunikatami, które są zapisywane w buforze odczytu z wyprzedzeniem, jeśli kolejka jest zamknięta.

- Użyj komendy MQCO_IMMEDIATE, aby usunąć komunikaty w buforze odczytu z wyprzedzeniem.
- Użyj komendy MQCO_QUIESCE, aby upewnić się, że komunikaty w buforze odczytu z wyprzedzeniem są konsumowane przez aplikację przed zamknięciem kolejki. Gdy zostanie wydana komenda MQCLOSE z opcją MQCO_QUIESCE, a w buforze odczytu z wyprzedzeniem znajdują się komunikaty, komenda MQRC_READ_AHEAD_MSGS powraca z parametrem MQCC_WARNING.

Strojenie wydajności dla nietrwałych komunikatów w systemie AIX
Jeśli używany jest produkt AIX V5.3 lub nowszy, należy rozważyć ustawienie parametru strojenia w celu użycia pełnej wydajności dla nietrwałych komunikatów.

Aby ustawić parametr strojenia w taki sposób, aby był on używany natychmiast, należy wprowadzić następującą komendę jako użytkownik root:

```
/usr/sbin/ioo -o j2_nPagesPerWriteBehindCluster=0
```

Aby ustawić parametr strojenia w taki sposób, aby był on uruchamiany natychmiast po restarcie, należy wprowadzić 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 program AIX może zaplanować zapisywanie nietrwałych komunikatów na dysku. Komunikaty zaplanowane do zapisania na dysku są niedostępne dla operacji MQGET, dopóki zapis na dysku nie zostanie zakończony. Sugerowana komenda strojenia zależy od tego progu. Zamiast zapisywania komunikatów w celu zapisania na dysku, gdy 16 kB danych znajduje się w kolejce, zapis na dysk występuje tylko wtedy, gdy rzeczywista pamięć masowa na komputerze staje się bliska zapełnieniowi. Jest to zmiana globalna i może mieć wpływ na inne komponenty oprogramowania.

W systemie AIX, gdy używane są aplikacje wielowątkowe, a w szczególności w przypadku uruchamiania na komputerach z wieloma procesorami, stanowczo zaleca się ustawienie wartości AIXTHREAD_SCOPE=S w identyfikatorze mqm `.profile` lub ustawienie wartości AIXTHREAD_SCOPE=S w środowisku przed uruchomieniem aplikacji, w celu zapewnienia lepszej wydajności i bardziej solidnego harmonogramu. Na przykład:

```
export AIXTHREAD_SCOPE=S
```

Ustawienie AIXTHREAD_SCOPE=S oznacza, że wątki użytkownika utworzone z atrybutami domyślnymi są umieszczane w zasięgu rywalizacji o zasięgu systemowym. Jeśli wątek użytkownika jest tworzony z zasięgiem rywalizacji o zasięgu systemowym, jest on powiązany z wątkiem jądra, który jest zaplanowany przez jądro. Bazowy wątek jądra nie jest współużytkowany z żadnym innym wątkiem użytkownika.

deskryptory plików.

W przypadku uruchamiania wielowątkowego procesu, takiego jak proces agenta, można osiągnąć miękki limit dla deskryptorów plików. This limit gives you the IBM MQ reason code `MQRC_UNEXPECTED_ERROR (2195)` and, if there are enough file descriptors, an IBM MQ FFST™ file.

Aby uniknąć tego problemu, można zwiększyć limit procesu dla liczby deskryptorów plików. W tym celu należy zmienić atrybut `nofiles` w `/etc/security/limits` na 10 000 w przypadku identyfikatora użytkownika mqm lub w sekcji default.

Limity zasobów systemowych

Ustaw limit zasobów systemowych dla segmentu danych i segmentu stosu na nieograniczony, korzystając z następujących komend w wierszu komend:

```
ulimit -d unlimited
ulimit -s unlimited
```

Typ indeksu

Atrybut kolejki *IndexType*określa typ indeksu utrzymanego przez menedżer kolejek w celu zwiększenia szybkości operacji MQGET w kolejce.

Uwaga: Obsługiwane tylko w systemie IBM MQ for z/OS.

Dostępne są pięć opcji:

Wartość	Opis
BRAK	Indeks nie jest obsługiwany. Użyj tej opcji podczas sekwencyjnego pobierania komunikatów (patrz “Priorytet” na stronie 865).

Wartość	Opis
groupID	Obsługiwany jest indeks identyfikatorów grup. Ten typ indeksu musi być używany, jeśli porządkowanie logiczne grup komunikatów ma być wyświetlane (patrz “Uporządkowanie logiczne i fizyczne” na stronie 866).
MSGID	Obsługiwany jest indeks identyfikatorów komunikatów. Użyj tej opcji podczas pobierania komunikatów przy użyciu pola *MsgId* jako kryterium wyboru w wywołaniu MQGET (patrz “Uzyskiwanie określonego komunikatu” na stronie 877).
MSGTOKEN,	Obsługiwany jest indeks tokenów komunikatów.
CORRELID	Obsługiwany jest indeks identyfikatorów korelacji. Użyj tej opcji podczas pobierania komunikatów przy użyciu pola *CorrelId* jako kryterium wyboru w wywołaniu MQGET (patrz “Uzyskiwanie określonego komunikatu” na stronie 877).

Uwaga:

1. W przypadku indeksowania za pomocą opcji MSGID lub opcji CORRELID należy ustawić względne parametry **MsgId** lub **CorrelId** w strukturze MQMD. Ustawienie obu tych opcji nie jest korzystne.
2. Podczas przeglądania używany jest mechanizm indeksowania w celu znalezienia komunikatu, jeśli kolejka jest zgodna z wszystkimi następującymi warunkami:
   - Ma typ indeksu MSGID, CORRELID lub GROUPID.
   - Jest on przeglądany z tym samym typem identyfikatora
   - Zawiera tylko komunikaty o jednym priorytecie.
3. Należy unikać kolejek (indeksowanych przez produkt *MsgId* lub *CorrelId*) zawierających tysiące komunikatów, ponieważ wpływa to na czas restartu. (Nie dotyczy to komunikatów nietrwałych, ponieważ są one usuwane przy restarcie).
4. Parametr MSGTOKEN jest używany do definiowania kolejek zarządzanych przez menedżera obciążenia produktu z/OS .

Pełny opis atrybutu **IndexType** znajduje się w sekcji IndexType. Więcej informacji na temat atrybutu **IndexType** zawiera sekcja “Uwagi dotyczące projektowania i wydajności dla aplikacji produktu z/OS” na stronie 66.

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 radzenia sobie z komunikatami, które są dłuższe niż 4 MB.

Można zwiększyć wartość atrybutu **MaxMsgLength** do 100 MB we wszystkich systemach IBM MQ w wersji V6 lub nowszej. Ustaw tę wartość, aby odzwierciedlić wielkość komunikatów korzystających z kolejki. W systemach IBM MQ innych niż IBM MQ for z/OSmożna również wykonać następujące czynności:

1. Użyj segmentowanych komunikatów. (Komunikaty mogą być segmentowane przez aplikację lub menedżer kolejek).
2. Użyj komunikatów odniesienia.

Każde z tych podejść zostało opisane w dalszej części niniejszej sekcji.

Zwiększanie maksymalnej długości komunikatu

Atrybut menedżera kolejek produktu **MaxMsgLength** definiuje maksymalną długość komunikatu, który może być obsługiwany przez menedżer kolejek. Podobnie, atrybut kolejki **MaxMsgLength** to maksymalna długość komunikatu, który może być obsługiwany przez kolejkę. Domyślna maksymalna obsługiwana długość komunikatu zależy od środowiska, w którym pracuje użytkownik.

W przypadku obsługi dużych komunikatów atrybuty te można zmieniać niezależnie na platformach innych niż z/OS. Wartość atrybutu menedżera kolejek można ustawić w zakresie od 32768 bajtów do 100 MB.

Ostrzeżenie: W systemie IBM MQ for z/OS atrybut **MaxMsgLength** menedżera kolejek jest zakodowany na stałe o wartości 100 MB.

Na wszystkich platformach można ustawić wartość atrybutu kolejki w zakresie od 0 do 100 MB.

Po zmianie jednego lub obu atrybutów programu **MaxMsgLength** zrestartuj aplikacje i kanały, aby zmiany zostały uwzględnione.

Po wprowadzeniu tych zmian długość komunikatu musi być mniejsza lub równa zarówno dla kolejki, jak i dla atrybutów **MaxMsgLength** menedżera kolejek. Jednak istniejące komunikaty mogą być dłuższe niż jeden z atrybutów.

Jeśli komunikat jest zbyt duży dla kolejki, zwracana jest wartość 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. Przed użyciem należy jednak rozważyć następujące czynniki:

- Równomierność wśród menedżerów kolejek jest zmniejszona. Maksymalna wielkość danych komunikatu jest określana przez *MaxMsgLength* dla każdej kolejki (włącznie z kolejkami transmisji), na której zostanie umieszczony komunikat. Ta wartość jest często ustawiana domyślnie na *MaxMsgLength*menedżera kolejek, zwłaszcza w przypadku kolejek transmisji. Utrudnia to określenie, czy komunikat jest zbyt duży, gdy ma być on podróżowany do zdalnego menedżera kolejek.
- Użycie zasobów systemowych zwiększa się. Na przykład aplikacje potrzebują większych buforów, a na niektórych platformach mogą być zwiększone użycie współużytkowanej pamięci masowej. Należy mieć wpływ na pamięć masową kolejki tylko wtedy, gdy jest to wymagane w przypadku większych komunikatów.
- Dotknięcie kanału jest zakłócające. Duży komunikat jest nadal liczony jako tylko jeden komunikat w stosunku do liczby partii, ale wymaga dłuższego przesyłania, zwiększając tym samym czasy odpowiedzi dla innych komunikatów.

Multi *Segmentacja komunikatów*

Ta sekcja zawiera informacje na temat segmentacji komunikatów. Ta funkcja nie jest obsługiwana w produkcie IBM MQ for z/OS ani przez aplikacje korzystające z produktu IBM MQ classes for JMS.

Zwiększenie maksymalnej długości komunikatu, jak wyjaśniono w temacie "Zwiększanie maksymalnej długości komunikatu" na stronie 883 , ma pewne negatywne konsekwencje. Ponadto nadal może spowodować, że komunikat jest zbyt duży dla kolejki lub menedżera kolejek. W takich przypadkach można segmentować komunikat. Informacje na temat segmentów zawiera sekcja "Grupy komunikatów" na stronie 44.

W następnych sekcjach używane są wspólne zastosowania służące do segmentacji komunikatów. Zakłada się, że wywołania MQPUT lub MQGET *zawsze* działają w ramach jednostki pracy. Zawsze warto rozważyć zastosowanie tej techniki w celu zmniejszenia możliwości obecnych w sieci grup niekompletnych. Zakłada się, że zatwierdzanie jednofazowe przez menedżer kolejek jest poprawne, ale inne techniki koordynacji są równie ważne.

Ponadto w aplikacjach pobierających przyjmuje się, że jeśli wiele serwerów przetwarza tę samą kolejkę, każdy serwer wykonuje podobny kod, tak aby jeden serwer nigdy nie znalazł komunikatu lub segmentu, który oczekiwał na to, że tam będzie (ponieważ podano wcześniej wartość MQGMO_ALL_MSGS_AVAILABLE lub MQGMO_ALL_SEGMENTS_AVAILABLE).

Umieszczanie i zdobycie posegmentowanego przekazu, który obejmuje jednostki pracy

Można umieścić i uzyskać segmentowany komunikat, który obejmuje jednostkę pracy w podobny sposób, jak w przypadku produktu "Umieszczanie i pobieranie grupy, która obejmuje jednostki pracy" na stronie 874.

Nie można jednak umieszczać lub umieszczać segmentowanych komunikatów w globalnej jednostce pracy.

Multi *Segmentacja i ponowne składanie przez menedżera kolejek*

Jest to najprostszy scenariusz, w którym jedna aplikacja umieszcza komunikat, który ma zostać pobrany przez inną aplikację. Komunikat może być duży: nie jest zbyt duży, aby można było obsłużyć aplikację lub aplikację pobierający w pojedynczym buforze, ale jest zbyt duży dla menedżera kolejek lub kolejki, w której ma zostać umieszczony komunikat.

Jedynymi zmianami niezbędnymi dla tych aplikacji są wprowadzanie aplikacji w celu autoryzowania menedżera kolejek w celu przeprowadzenia segmentacji, 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 aplikacji pobierających, aby poprosić menedżera kolejek 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, tak aby menedżer kolejek mógł wygenerować unikalny identyfikator grupy dla każdego komunikatu. Jeśli nie jest to zrobione, niepowiązane komunikaty mogą mieć ten sam identyfikator grupy, co może prowadzić do nieprawidłowego przetwarzania.

Bufor aplikacji musi być na tyle duży, aby zawierał ponownie zmontowany komunikat (chyba że zostanie podana opcja MQGMO_ACCEPT_TRUNCATED_MSG).

Jeśli atrybut MAXMSGLEN kolejki ma być modyfikowany w taki sposób, aby pomieścić segmentację komunikatów, należy rozważyć następujące kwestie:

- Minimalny segment komunikatu obsługiwany w kolejce lokalnej to 16 bajtów.
- W przypadku kolejki transmisji wartość MAXMSGLEN musi zawierać również miejsce wymagane dla nagłówków. Należy rozważyć użycie wartości o wielkości co najmniej 4000 bajtów większej od maksymalnej oczekiwanej długości danych użytkownika w dowolnym segmencie komunikatów, który można umieścić w kolejce transmisji.

Jeśli konieczna jest konwersja danych, może to być konieczne, aby aplikacja pobierała wartość MQGMO_CONVERT. Powinno to być proste, ponieważ wyjście konwersji danych jest przedstawiane wraz z kompletnym komunikatem. Nie próbuj konwertować danych w kanale nadawczym, jeśli wiadomość jest posegmentowana, 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 jest niewystarczająca lub gdy aplikacje wymagają konwersji danych z określonymi granicami segmentów.

Segmentacja aplikacji jest używana z dwóch głównych powodów:

1. Sama segmentacja menedżera kolejek jest niewystarczająca, ponieważ komunikat jest zbyt duży, aby można go było obsłużyć w jednym buforze przez aplikacje.
2. Konwersja danych musi być wykonywana przez kanały nadawcze, a format jest taki, że aplikacja musi określać miejsce, w którym granice segmentu mają być w celu przekształcenia danego segmentu w możliwy sposób.

Jeśli jednak konwersja danych nie jest problemem lub jeśli aplikacja pobierający zawsze używa komendy MQGMO_COMPLETE_MSG, segmentacja menedżera kolejek może być również dozwolona przez

określenie parametru MQMF_SEGMENTATION_ALLOWED. W naszym przykładzie aplikacja segmentuje komunikat w czterech segmentach:

```
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 zostanie użyty, aplikacja musi ustawić *Offset* i długość każdego segmentu. W takim przypadku stan logiczny nie jest obsługiwany automatycznie.

Aplikacja pobierające nie może zagwarantować, że bufor jest wystarczająco duży, aby pomieścić wszystkie ponownie złożone komunikaty. W związku z tym musi być ona przygotowana do indywidualnego przetwarzania segmentów.

W przypadku komunikatów posegmentowanych ta aplikacja nie chce rozpoczynać przetwarzania jednego segmentu, dopóki wszystkie segmenty, które nie stanowią komunikatu logicznego, są obecne. Wartość MQGMO_ALL_SEGMENTS_AVAILABLE jest zatem określona dla pierwszego segmentu. Jeśli określono parametr MQGMO_LOGICAL_ORDER, a istnieje bieżący komunikat logiczny, opcja MQGMO_ALL_SEGMENTS_AVAILABLE jest ignorowana.

Po pobraniu pierwszego segmentu komunikatu logicznego należy użyć komendy MQGMO_LOGICAL_ORDER, aby upewnić się, że pozostałe segmenty komunikatu logicznego są pobierane w kolejności.

Wiadomości nie są brane pod uwagę w różnych grupach. Jeśli takie komunikaty wystąpią, są one przetwarzane w kolejności, w jakiej pierwszy segment każdego komunikatu występuje 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 komunikatów logicznych*

Komunikaty muszą być przechowywane 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 komunikatów logicznych. Wszystkie, ale trzeci komunikat są duże i wymagają segmentacji, która jest wykonywana przez umieszczanie aplikacji:

```
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 pobierających wartość MQGMO_ALL_MSGS_AVAILABLE jest określona w pierwszej operacji MQGET. Oznacza to, że nie są pobierane żadne komunikaty ani segmenty grupy, dopóki cała grupa nie

będzie dostępna. Po pobraniu pierwszej fizycznej wiadomości grupy MQGMO_LOGICAL_ORDER służy do 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, komenda MQGMO_ALL_MSGS_AVAILABLE zostanie zignorowana.

Komunikaty odniesienia
Te informacje umożliwiają zapoznanie się z informacjami na temat komunikatów referencyjnych.

Uwaga: Nieobsługiwane w produkcie IBM MQ for z/OS.

Ta metoda umożliwia przeniesienie dużego obiektu z jednego węzła do drugiego bez zapisywania obiektu w kolejkach produktu IBM MQ w węźle źródłowym lub docelowym. Jest to szczególnie korzystne, gdy dane istnieją w innej formie, na przykład w przypadku aplikacji poczty elektronicznej.

W tym celu należy określić wyjście komunikatu na obu końcach kanału. Informacje na temat sposobu wykonania tej czynności zawiera sekcja “Programy obsługi wyjścia komunikatów kanału” na stronie 1078.

IBM MQ definiuje format nagłówka komunikatu odwołania (MQRMH). Opis tego parametru zawiera sekcja MQRMH . Jest on rozpoznawany przy użyciu zdefiniowanej nazwy formatu i może po niej następować rzeczywiste dane.

Aby zainicjować przesyłanie dużego obiektu, aplikacja może umieścić komunikat składający się z nagłówka komunikatu odwołania bez następującego po nim danych. Ponieważ ten komunikat opuszcza węzeł, wyjście komunikatu pobiera obiekt w odpowiedni sposób i dodaje go do komunikatu odniesienia. Następnie zwraca komunikat (obecnie większy niż poprzednio) do wysyłającego agenta kanału komunikatów w celu przesłania do odbierającego agenta MCA.

Inne wyjście komunikatów jest skonfigurowane w odbierającym MCA. Gdy to wyjście komunikatu odbiera jeden z tych komunikatów, tworzy obiekt przy użyciu danych obiektu, które zostały dopisane i przekazuje komunikat *bez* tego komunikatu. Komunikat odniesienia może teraz zostać odebrany przez aplikację, a aplikacja wie, że obiekt (lub co najmniej jego część reprezentowana przez ten komunikat odniesienia) został utworzony w tym węźle.

Maksymalna ilość danych obiektu, które program zewnętrzny wysyłający komunikat może dopisać do komunikatu referencyjnego, jest ograniczona przez wynegocjowaną maksymalną długość komunikatu dla kanału. Wyjście może zwrócić tylko jeden komunikat do agenta MCA dla każdego komunikatu, który został przekazany, tak więc aplikacja umieszczanie może umieścić kilka komunikatów, aby spowodować przeniesienie jednego obiektu. Każdy komunikat musi identyfikować *logiczną* długość i przesunięcie obiektu, który ma być do niego dodawany. Jednak w przypadkach, gdy nie jest możliwe poznanie łącznej wielkości obiektu lub maksymalnej wielkości dozwolonej przez kanał, zaprojektuj wyjście komunikatu, tak aby aplikacja umieszczała tylko jeden komunikat, a samo wyjście umieszcza następny komunikat w kolejce transmisji, gdy dopisał on tyle danych, ile może do wiadomości, które zostało przekazane.

Przed skorzystaniem z tej metody postępowania z dużymi komunikatami należy rozważyć następujące kwestie:

- Agent MCA i wyjście komunikatów są uruchamiane pod ID użytkownika IBM MQ . Wyjście komunikatu (a więc identyfikator użytkownika) wymaga dostępu do obiektu w celu pobrania go na końcu wysyłającego lub utworzenia go na końcu odbierającym. Może to być możliwe tylko w przypadkach, gdy obiekt jest powszechnie dostępny. To rodzi problem z bezpieczeństwem.

- Jeśli komunikat referencyjny z dołączonymi danymi masowymi musi przejść przez kilka menedżerów kolejek przed dotarciem do miejsca docelowego, dane masowe są obecne w kolejkach produktu IBM MQ w węzłach, które są w tym odstępie czasu. W takich przypadkach nie trzeba jednak zapewnić specjalnego wsparcia lub wyjścia.
- Projektowanie wyjścia komunikatów jest utrudnione, jeśli dozwolone jest kolejkowanie przekierowywania lub kolejkowania niewysłanych wiadomości. W takich przypadkach fragmenty obiektu mogą być odbierane poza kolejką.
- Gdy komunikat odniesienia dociera do miejsca docelowego, wyjście komunikatu odbierającego powoduje utworzenie obiektu. Nie jest to jednak synchronizowane z jednostką pracy agenta MCA, więc jeśli partia jest wycofana, w późniejszym zadaniu wsadowym pojawi się inny komunikat referencyjny zawierający tę samą część obiektu, a wyjście komunikatu może próbować ponownie utworzyć tę samą część obiektu. Jeśli obiekt jest na przykład serią aktualizacji bazy danych, może to być niedopuszczalne. Jeśli tak, wyjście komunikatu musi zawierać dziennik, którego aktualizacje zostały zastosowane. Może to wymagać użycia kolejki produktu IBM MQ .
- W zależności od charakterystyki typu obiektu, wyjścia komunikatów i aplikacje mogą wymagać współpracy przy obsłudze liczników użycia, aby można było usunąć obiekt, gdy nie jest on już potrzebny. Być może także wymagany jest identyfikator instancji; w nagłówku komunikatu referencyjnego jest dostępne pole (patrz MQRMH).
- Jeśli komunikat odniesienia jest umieszczany jako lista dystrybucyjna, obiekt musi być pobieralny dla każdej wynikowej listy dystrybucyjnej lub pojedynczego miejsca docelowego w tym węźle. Może być konieczne utrzymanie liczników użycia. Należy również rozważyć możliwość, że węzeł może być węzłem końcowym dla niektórych miejsc docelowych na liście, ale dla innych węzłów pośrednich.
- Dane masowe nie są zazwyczaj przekształcane. Jest to spowodowane tym, że konwersja ma miejsce *przed* wywołaniem wyjścia komunikatu. Z tego powodu konwersja nie może być żądana dla źródłowego kanału nadawczego. Jeśli komunikat odniesienia przechodzi przez węzeł pośredni, dane masowe są przekształcane po wysłaniu z węzła pośredniego, jeśli jest to wymagane.
- Nie można segmentować komunikatów referencyjnych.

Korzystanie z struktur MQRMH i MQMD

Patrz MQRMH i MQMD , aby uzyskać opis pól w nagłówku komunikatu odwołania i w deskryptorze komunikatu.

W strukturze MQMD ustaw pole *Format* na wartość MQFMT_REF_MSG_HEADER. Format MQHREF, jeśli jest wymagany dla komendy MQGET, jest automatycznie przekształcany przez program IBM MQ wraz z następującymi danymi masowymi, które są następujące.

Poniżej znajduje się przykład użycia pól *DataLogicalOffset* i *DataLogicalLength* w MQRMH:

Aplikacja umieszczana w aplikacji może umieścić komunikat referencyjny z:

- Brak danych fizycznych
- *DataLogicalLength* = 0 (ten komunikat reprezentuje cały obiekt)
- *DataLogicalOffset* = 0.

Zakładając, że obiekt ma długość 70 000 bajtów, wyjście komunikatu wysyłającego wysyła pierwsze 40 000 bajtów wzdłuż kanału w komunikacie referencyjnym zawierającym:

- 40 000 bajtów danych fizycznych po MQRMH
- *DataLogicalLength* = 40000
- *DataLogicalOffset* = 0 (od początku obiektu).

Następnie umieszcza inny komunikat w kolejce transmisji zawierającej:

- Brak danych fizycznych
- *DataLogicalLength* = 0 (do końca obiektu). Można tu podać wartość 30 000.
- *DataLogicalOffset* = 40000 (począwszy od tego punktu).

Gdy wyjście komunikatu jest widoczne przez wyjście 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).

Ustawiona jest również flaga MQRMHF_LAST.

Opis przykładowych programów udostępnionych w celu użycia komunikatów referencyjnych znajduje się w sekcji "Korzystanie z przykładowych programów na platformie Multiplatforms" na stronie 1166.

Oczekiwanie na komunikaty

Jeśli program ma czekać do momentu nadejścia komunikatu do kolejki, należy podać opcję MQGMO_WAIT w polu *Options* struktury MQGMO.

Użyj pola *WaitInterval* w strukturze MQGMO, aby określić Maksymalny czas (w milisekundach), przez jaki wywołanie MQGET ma oczekiwać na przesłanie komunikatu do kolejki.

Jeśli komunikat nie zostanie wyświetlony w tym czasie, wywołanie MQGET zostanie zakończone z kodem przyczyny MQRC_NO_MSG_AVAILABLE.

W polu *WaitInterval* można określić nieograniczony przedział czasu oczekiwania, korzystając ze stałej MQWI_UNLIMITED. Jednak zdarzenia znajdujące się poza kontrolą mogą spowodować, że program będzie czekał przez długi czas, dlatego należy zachować ostrożność przy zachowaniu ostrożności. Aplikacje IMS nie mogą określać nieograniczonego przedziału czasu oczekiwania, ponieważ uniemożliwiłoby to zakończenie działania systemu IMS . (Gdy program IMS kończy działanie, konieczne jest zakończenie wszystkich zależnych od niego regionów). Zamiast tego aplikacje IMS mogą określać skończony okres oczekiwania, a następnie, jeśli wywołanie zakończy się bez pobierania komunikatu po upływie tego okresu, wywołaj inną wywołanie 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 przylot komunikatu. Jeśli jednak więcej niż jeden program oczekuje na przeglądanie wiadomości, wszystkie programy mogą być aktywowane. Więcej informacji na ten temat zawiera opis pola *Options* w strukturze MQGMO w produkcie MQGMO.

Jeśli stan kolejki lub menedżer kolejek ulegnie zmianie przed upływem odstępu czasu oczekiwania, wykonywane są następujące działania:

- Jeśli menedżer kolejek przejdzie w stan wygaszania, a użyto opcji MQGMO_FAIL_IF_QUIESCING, to oczekiwanie zostanie anulowane, a wywołanie MQGET zakończy się z kodem przyczyny MQRC_Q_MGR_QUIESCING. Bez tej opcji połączenie oczekuje na oczekiwanie.
- z/OS W systemie z/OS, jeśli połączenie (dla aplikacji CICS lub IMS) wchodzi w stan wygaszania, a użyto opcji MQGMO_FAIL_IF_QUIESCING, oczekiwanie zostanie anulowane, a wywołanie MQGET zakończy się z kodem przyczyny MQRC_CONN_QUIESCING. Bez tej opcji połączenie oczekuje na oczekiwanie.
- Jeśli menedżer kolejek jest zmuszony do zatrzymania lub został anulowany, wywołanie MQGET kończy się albo z kodem przyczyny MQRC_Q_MGR_ZATRZYMYWANIA, albo z kodem przyczyny MQRC_CONNECTION_BROKEN.
- Jeśli atrybuty kolejki (lub kolejki, do której tłumaczona jest nazwa kolejki) zostaną zmienione w taki sposób, że żądania pobierania są teraz wstrzymane, oczekiwanie zostanie anulowane, a wywołanie MQGET zakończy się z kodem przyczyny MQRC_GET_INHIBITED.
- Jeśli atrybuty kolejki (lub kolejki, do której tłumaczona jest nazwa kolejki), zostaną zmienione w taki sposób, że wymagana jest opcja FORCE, to oczekiwanie zostanie anulowane, a wywołanie MQGET zakończy się kodem przyczyny MQRC_OBJECT_CHANGED.

z/OS Jeśli aplikacja ma czekać na więcej niż jedną kolejkę, należy użyć funkcji sygnalizacji IBM MQ for z/OS (patrz "sygnalizowanie" na stronie 890). Więcej informacji na temat okoliczności, w których występują te działania, zawiera sekcja MQGMO.

sygnalizowanie

Sygnalizacja jest obsługiwana tylko w systemie IBM MQ for z/OS.

Sygnalizacja jest opcją w wywołaniu MQGET, aby umożliwić systemowi operacyjnie powiadamianie (lub *sygnał*) Program, w którym oczekiwany komunikat pojawia się w kolejce. Jest to działanie podobne do funkcji *get with wait* opisanej w temacie "Oczekiwanie na komunikaty" na stronie 889 , ponieważ umożliwia on programowi kontynuowanie pracy z innymi pracami podczas oczekiwania na sygnał. Jeśli jednak używane jest sygnalizowanie, można zwolnić wątek aplikacji i polegać na systemie operacyjnym, aby powiadomić program po nadejściu komunikatu.

Aby ustawić sygnał

Aby ustawić sygnał, wykonaj następujące czynności w strukturze MQGMO, która jest używana w wywołaniu MQGET:

1. Ustaw opcję MQGMO_SET_SIGNAL w polu *Options* .
2. Ustaw maksymalny czas życia sygnału w polu *WaitInterval* . Ustawia czas (w milisekundach), przez jaki program IBM MQ ma monitorować kolejkę. Aby określić nieograniczone życie, należy użyć wartości MQWI_UNLIMITED.

   Uwaga: Aplikacje IMS nie mogą określać nieograniczonego przedziału czasu oczekiwania, ponieważ uniemożliwiłoby to zakończenie działania systemu IMS . (Gdy program IMS kończy działanie, konieczne jest zakończenie wszystkich zależnych od niego regionów). Zamiast tego aplikacje IMS mogą w regularnych odstępach czasu sprawdzać stan EBC (patrz krok 3). W programie mogą być jednocześnie ustawione sygnały w kilku uchwytach kolejki:
3. W polu *Signal1* podaj adres *bloku kontrolnego zdarzeń* (ECB). Spowoduje to powiadomienie użytkownika o wyniku sygnału. Pamięć masowa EBC musi pozostać dostępna do czasu zamknięcia kolejki.

Uwaga: Z opcją MQGMO_WAIT nie można używać opcji MQGMO_SET_SIGNAL.

Po nadejściu komunikatu

Po nadejściu odpowiedniego komunikatu do EBC zwracany jest kod zakończenia.

Kod zakończenia opisuje jedną z następujących wartości:

- Komunikat, dla którego ustawiono sygnał dla kolejki, został wysłany do kolejki. Komunikat nie jest zarezerwowany dla programu, który zażądał sygnału, więc program musi wywołać wywołanie MQGET ponownie w celu pobrania komunikatu.

  Uwaga: Inna aplikacja może pobrać komunikat w czasie między odbierającym sygnał i wywołaniem innego wywołania MQGET.
- Ustawiony odstęp czasu oczekiwania utracił ważność, a komunikat, dla którego został ustawiony sygnał, nie dotarł do kolejki. Program IBM MQ anulował sygnał.
- Sygnał został anulowany. Zdarza się to na przykład wtedy, gdy menedżer kolejek zostanie zatrzymany lub atrybut kolejki zostanie zmieniony, tak aby wywołania MQGET nie były już dozwolone.

Gdy odpowiedni komunikat znajduje się już w kolejce, wywołanie MQGET jest wykonywane w taki sam sposób, jak wywołanie MQGET bez sygnalizowania. Ponadto, jeśli błąd zostanie wykryty natychmiast, wywołanie zostanie zakończone, a kody powrotu są ustawione.

Gdy połączenie jest akceptowane i żaden komunikat nie jest natychmiast dostępny, sterowanie jest zwracane do programu, tak aby można było kontynuować pracę z innymi. Żaden z pól wyjściowych w deskryptorze komunikatu nie jest ustawiony, ale parametr **CompCode** jest ustawiony na wartość MQCC_WARNING, a parametr **Reason** ma wartość MQRC_SIGNAL_REQUEST_ACCEPTED.

Informacje o tym, co IBM MQ może powrócić do aplikacji, gdy wywołują wywołanie MQGET przy użyciu sygnalizowania, zawiera sekcja MQGET.

Jeśli program nie ma innej pracy do wykonania, gdy oczekuje na opublikowanie EBC, może on czekać na EBC, korzystając z:

- 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 menedżer kolejek ulegnie zmianie w czasie, gdy sygnał jest ustawiony (to znaczy, że EBC nie został jeszcze opublikowany), wykonywane są następujące działania:

- Jeśli menedżer kolejek przejdzie do stanu wygaszania, a użyto opcji MQGMO_FAIL_IF_QUIESCING, to sygnał jest anulowany. EBC jest księgowany za pomocą kodu zakończenia MQEC_Q_MGR_QUIESCING. Bez tej opcji sygnał pozostaje ustawiony.
- Jeśli menedżer kolejek jest zmuszony do zatrzymania lub został anulowany, sygnał jest anulowany. Sygnał jest dostarczany wraz z kodem zakończenia MQEC_WAIT_ANULOWANY.
- Jeśli atrybuty kolejki (lub kolejki, do której jest rozstrzygana nazwa kolejki) zostaną zmienione, tak że żądania pobierania są teraz zablokowane, sygnał jest anulowany. Sygnał jest dostarczany wraz z kodem zakończenia MQEC_WAIT_ANULOWANY.

Uwaga:

1. Jeśli więcej niż jeden program ustawił sygnał w tej samej współużytkowanej kolejce w celu usunięcia komunikatu, tylko jeden program jest aktywowany przez przylot komunikatu. Jeśli jednak więcej niż jeden program oczekuje na przeglądanie wiadomości, wszystkie programy mogą być aktywowane. Reguły, które menedżer kolejek podąża za podjęciem decyzji o aktywowaniu aplikacji, 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 MQGMO-Get-message).
2. Jeśli istnieje więcej niż jedno wywołanie MQGET dla tego samego komunikatu, z mieszaniną opcji oczekiwania i sygnału, każde oczekujące wywołanie jest traktowane jednakowo. Więcej informacji na ten temat zawiera opis pola *Options* struktury MQGMO w sekcji MQGMO-Get-message options(Opcje MQGMO-Get-message).
3. W pewnych warunkach możliwe jest zarówno wywołanie MQGET w celu pobrania komunikatu, jak i sygnał (wynikający z przybycia tego samego komunikatu), który ma zostać dostarczony. Oznacza to, że w przypadku wydania przez program innego wywołania MQGET (ponieważ sygnał został dostarczony), komunikat nie może być dostępny. Zaprojektuj program, aby przetestować tę sytuację.

Informacje na temat sposobu ustawiania sygnału można znaleźć w opisie opcji MQGMO_SET_SIGNAL i w polu *Signal1* w pliku Signal1.

Pomijanie wycofania

Aby program użytkowy nie mógł wejść do pętli *MQGET-error-backout* , można określić opcję **MQGMO_MARK_SKIP_BACKOUT** w wywołaniu MQGET.

Uwaga: Obsługiwane tylko w systemie IBM MQ for z/OS.

W ramach jednostki pracy program użytkowy może wydać jedną lub więcej wywołań MQGET w celu pobrania komunikatów z kolejki. Jeśli program użytkowy wykryje błąd, może wycofać się z jednostki pracy. Spowoduje to odtworzenie wszystkich zasobów zaktualizowanych podczas tej jednostki pracy do stanu, w którym znajdowały się one przed uruchomieniem jednostki pracy, a następnie przywróci komunikaty pobrane przez wywołania MQGET.

Po przywróceniu te komunikaty są dostępne dla kolejnych wywołań MQGET wydawanych przez program użytkowy. W wielu przypadkach nie powoduje to problemu z programem użytkowym. Jednak w przypadku, gdy błąd prowadzący do wycofania nie może być obejście, komunikat przywrócony do kolejki może spowodować, że program użytkowy wprowadzi pętlę *MQGET-error-backout* .

Aby uniknąć tego problemu, należy określić opcję MQGMO_MARK_SKIP_BACKOUT w wywołaniu MQGET. Oznacza to, że żądanie MQGET nie jest używane w wycofaniu zainicjowanym przez aplikację. To znaczy, że nie może on być wycofany. Użycie tej opcji oznacza, że w razie wystąpienia wycofania aktualizacje innych zasobów są wycofane zgodnie z wymaganiami, ale oznaczony komunikat jest traktowany tak, jakby został pobrany w ramach nowej jednostki pracy.

Program użytkowy musi wywołać wywołanie IBM MQ w celu zatwierdzenia nowej jednostki pracy lub na potrzeby utworzenia kopii zapasowej nowej jednostki pracy. Na przykład program może wykonać obsługę wyjątków, na przykład informując inicjatora, że komunikat został usunięty, i zatwierdzić jednostkę pracy,

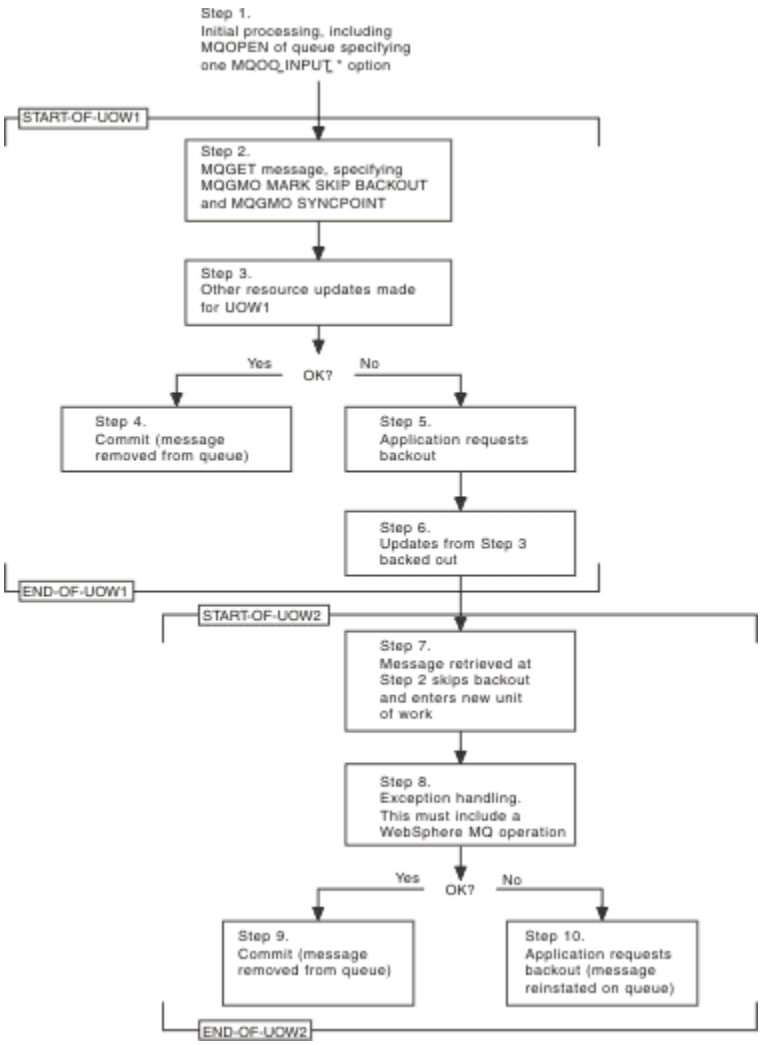
tak aby usunąć komunikat z kolejki. Jeśli nowa jednostka pracy zostanie wycofana (z jakiegokolwiek powodu), komunikat zostanie przywrócony do kolejki.

W ramach jednostki pracy może istnieć tylko jedno żądanie MQGET oznaczone jako pominięcie wycofania. Jednak może istnieć kilka innych komunikatów, które nie są oznaczone jako pomijanie wycofania. Gdy komunikat zostanie oznaczony jako pomijanie wycofania, wszystkie kolejne wywołania MQGET w ramach jednostki pracy, które określają wartość MQGMO_MARK_SKIP_BACKOUT, kończą się niepowodzeniem. Kod przyczyny: MQRC_SECOND_MARK_NOT_ALLOWED.

Uwaga:

1. Zaznaczony komunikat pomija wycofany komunikat tylko wtedy, gdy jednostka pracy zawierająca ją zostanie zakończona przez żądanie aplikacji w celu wycofania go. Jeśli jednostka pracy zostanie wycofana z innego powodu, zostanie ona wycofana do kolejki w taki sam sposób, w jaki będzie miała miejsce, gdyby nie została oznaczona do pominięcia wycofania.
2. Pominięcie wycofania nie jest obsługiwane w ramach procedur składowanych Db2 uczestniczących w jednostkach pracy kontrolowanych przez usługi RRS. Na przykład wywołanie MQGET z opcją MQGMO_MARK_SKIP_BACKOUT nie powiedzie się i zostanie zwrócony kod przyczyny MQRC_OPTION_ENVIRONMENT_ERROR.

Rysunek 68 na stronie 892 przedstawia typową sekwencję kroków, które może zawierać program użytkowy, gdy żądanie MQGET jest wymagane do pominięcia wycofania.



Rysunek 68. Pomijanie wycofań za pomocą komendy MQGMO_MARK_SKIP_BACKOUT

Kroki w programie Rysunek 68 na stronie 892 są następujące:

Krok 1
Początkowe przetwarzanie odbywa się w ramach transakcji, w tym wywołanie MQOPEN w celu otwarcia kolejki (podanie jednej z opcji MQOO_INPUT_ * w celu pobrania komunikatów z kolejki w kroku 2).

Krok 2
Wywołano komendę MQGET z opcją MQGMO_SYNCPOINT i MQGMO_MARK_SKIP_BACKOUT. MQGMO_SYNCPOINT jest wymagany, ponieważ MQGET musi znajdować się w jednostce pracy dla komendy MQGMO_MARK_SKIP_BACKOUT, aby była ona efektywna. W produkcie Rysunek 68 na stronie 892 ta jednostka pracy jest określana jako UOW1.

Krok 3
Inne aktualizacje zasobów są wykonywane jako część UOW1. Mogą one obejmować kolejne wywołania MQGET (wydane bez komendy MQGMO_MARK_SKIP_BACKOUT).

Krok 4
Wszystkie aktualizacje z kroków 2 i 3 są kompletne zgodnie z wymaganiami. Program użytkowy zatwierdza aktualizacje, a UOW1 kończy działanie. Komunikat pobrany w kroku 2 jest usuwany z kolejki.

Krok 5
Niektóre aktualizacje z kroków 2 i 3 nie są kompletne zgodnie z wymaganiami. Program użytkowy żąda, aby aktualizacje dokonane podczas wykonywania tych kroków zostały wycofane.

Krok 6
Aktualizacje wprowadzone w kroku 3 są 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 wycofany element UOW1 . (Na przykład wywołanie MQPUT w innej kolejce, wskazując, że wystąpił problem, który spowodował, że UOW1 został wycofany).

Krok 9
Krok 8 zakończy się zgodnie z wymaganiami, program aplikacji zatwierdza działanie, a UOW2 kończy działanie. Ponieważ żądanie MQGET jest częścią obiektu UOW2 (patrz krok 7), to zatwierdzenie powoduje usunięcie komunikatu z kolejki.

Krok 10
Krok 8 nie jest kompletny, jeśli jest wymagany, a program użytkowy tworzy kopię zapasową UOW2. Ponieważ żądanie pobrania komunikatu jest częścią UOW2 (patrz krok 7), to jest on również wycofany i przywrócony do kolejki. Jest on teraz dostępny dla kolejnych wywołań MQGET wydanych przez ten lub inny program użytkowy (w taki sam sposób, jak każdy inny komunikat w kolejce).

Konwersja danych aplikacji

Jeśli to konieczne, MCAs przekształci deskryptor komunikatu i dane nagłówka w wymagany zestaw znaków i kodowanie. Konwersja może być zakończona albo na końcu łącza (czyli lokalnego agenta MCA lub zdalnego agenta MCA).

Gdy aplikacja umieszcza komunikaty w kolejce, lokalny menedżer kolejek dodaje informacje sterujące do deskryptorów komunikatów w celu ułatwienia sterowania komunikatami podczas ich przetwarzania przez menedżery kolejek i MCAs. W zależności od środowiska, pola danych nagłówka komunikatu są tworzone w zestawie znaków i kodowaniu systemu lokalnego.

W przypadku przenoszenia komunikatów między systemami czasami zachodzi konieczność przekształcenia danych aplikacji w zestaw znaków i kodowanie wymagane przez system odbierający. Można to zrobić albo z poziomu programów aplikacji w systemie odbierającym, albo przez MCAs w systemie wysyłającym. Jeśli konwersja danych jest obsługiwana w systemie odbierającym, należy użyć programów aplikacji w celu przekształcenia danych aplikacji, a nie w zależności od konwersji, która już wystąpiła w systemie wysyłającym.

Dane aplikacji są przekształcane w program użytkowy po określeniu opcji MQGMO_CONVERT w polu *Options* struktury MQGMO przekazanej do wywołania MQGET, a *wszystkie* następujące instrukcje są prawdziwe:

- Pola *CodedCharSetId* lub *Encoding* ustawione w strukturze MQMD powiązanej z komunikatem w kolejce różnią się w zależności 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* podana w wywołaniu MQGET nie jest równa zero.
- Długość danych komunikatu nie jest równa zero.
- 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 kodowanego zestawu znaków i kodowania maszynowego 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 ten komunikat. Jeśli *Format* nie jest jednym z wbudowanych formatów, należy napisać wyjście konwersji danych, aby przekształcić ten komunikat.

Jeśli wysyłającym MCA jest przekształcanie danych, należy określić słowo kluczowe CONVERT (YES) w definicji każdego kanału nadawczego lub serwera, dla którego wymagana jest konwersja. Jeśli konwersja danych nie powiedzie się, komunikat jest wysyłany do kolejki DLQ w wysyłającym menedżerze kolejek, a pole *Feedback* struktury MQDLH wskazuje przyczynę. Jeśli komunikat nie może zostać umieszczony w kolejce DLQ, kanał zostanie zamknięty, a nieprzekształcony komunikat pozostaje w kolejce transmisji. Konwersja danych w aplikacjach, a nie przy wysyłaniu MCAs pozwala uniknąć tej sytuacji.

Z reguły dane w komunikacie opisanym jako dane *znakowe* przez wbudowany format lub wyjście konwersji danych są konwertowane z kodowanego zestawu znaków używanego przez komunikat do tego żądania, a pola *liczbowe* są przekształcane na żądane kodowanie.

Więcej szczegółowych informacji na temat konwencji przetwarzania konwersji używanych podczas przekształcania wbudowanych formatów oraz informacje na temat pisania własnych wyjść konwersji danych zawiera sekcja “Pisanie wyjść konwersji danych” na stronie 1082. Więcej informacji na temat tabel obsługi języków i obsługiwanych kodowań maszyn można znaleźć w sekcji Języki narodowe i Kodowanie maszyn .

Konwersja znaków nowego wiersza EBCDIC

Jeśli zachodzi potrzeba zapewnienia, że dane wysyłane z platformy EBCDIC do formatu ASCII są identyczne z danymi otrzymanego z powrotem, należy kontrolować konwersję znaków nowego wiersza EBCDIC.

Można to zrobić za pomocą przełącznika zależnego od platformy, który wymusza produkt IBM MQ do używania niezmodyfikowanych tabel konwersji, ale musi być świadomy niespójnego zachowania, które może spowodować.

Problem pojawia się, ponieważ znak nowego wiersza EBCDIC nie jest konwertowany w sposób spójny przez platformy lub tabele konwersji. W wyniku tego, jeśli dane są wyświetlane na platformie ASCII, formatowanie może być niepoprawne. Może to utrudnić, na przykład, administrowanie systemem IBM i zdalnie z poziomu platformy ASCII za pomocą komendy RUNMQSC.

Więcej informacji na temat konwersji danych w formacie EBCDIC na format ASCII zawiera sekcja Konwersja danych .

Przeglądanie komunikatów w kolejce

W tej sekcji znajdują się informacje na temat przeglądania komunikatów w kolejce przy użyciu wywołania MQGET.

Aby skorzystać z wywołania MQGET w celu przeglądania komunikatów w kolejce:

1. Wywołaj komendę MQOPEN, aby otworzyć kolejkę do przeglądania, określając opcję MQOO_BROWSE.
2. Aby przejrzeć pierwszy komunikat w kolejce, wywołaj komendę MQGET z opcją MQGMO_BROWSE_FIRST. Aby znaleźć odpowiedni komunikat, wywołaj komendę MQGET wielokrotnie z opcją MQGMO_BROWSE_NEXT, aby przejść przez wiele komunikatów.

   Po każdym wywołaniu MQGET w celu wyświetlenia wszystkich komunikatów konieczne jest ustawienie pól *MsgId* i *CorrelId* struktury MQMD na wartość NULL.
3. Wywołaj komendę MQCLOSE, aby zamknąć kolejkę.

Kursor przeglądania
Po otwarciu (MQOPEN) w kolejce do przeglądania, wywołanie tworzy kursor przeglądania do użycia z wywołaniami MQGET, które korzystają z jednej z opcji przeglądania. Kursor przeglądania jest wyświetlany jako wskaźnik logiczny, który znajduje się przed pierwszym komunikatem w kolejce.

Istnieje możliwość użycia więcej niż jednego kursora przeglądania (z jednego programu) przez wydanie kilku 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 spełniającą warunki określone w strukturze MQMD.

MQGMO_BROWSE_NEXT
Pobiera kopię następnego komunikatu spełniającą warunki określone w strukturze MQMD.

MQGMO_BROWSE_MSG_UNDER_CURSOR
Pobiera kopię komunikatu, który jest aktualnie wskazywał przez kursor, czyli ten, 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 pozycjonowany logicznie tuż przed pierwszym komunikatem w kolejce. Oznacza to, że jeśli wywołanie MQGET zostanie natychmiast po wywołaniu MQOPEN, można skorzystać z opcji MQGMO_BROWSE_NEXT w celu przeglądania pierwszego komunikatu. Nie trzeba używać 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 865.)

- “Kolejki w sekwencji FIFO (pierwszy raz w pierwszej kolejności)” na stronie 895
- “Kolejki w kolejności priorytetów” na stronie 895
- “Niezatwierdzone komunikaty” na stronie 896
- “Zmiana kolejności kolejek” na stronie 896
- “Korzystanie z indeksu kolejki” na stronie 896

Kolejki w sekwencji FIFO (pierwszy raz w pierwszej kolejności)

Pierwszym komunikatem w kolejce w tej sekwencji jest komunikat, który był w kolejce najdłuższy.

Użyj komendy MQGMO_BROWSE_NEXT, aby odczytać komunikaty sekwencyjnie w kolejce. Podczas przeglądania wyświetlane są wszystkie komunikaty umieszczone w kolejce, ponieważ kolejka w tej sekwencji ma komunikaty umieszczone na końcu. Gdy kursor rozpozna, że dotarł do końca kolejki, kursor przeglądania pozostaje w miejscu, w którym jest i powraca z parametrem MQRC_NO_MSG_AVAILABLE. Następnie można je pozostawić w oczekiwaniu na dalsze komunikaty lub zresetować na początku kolejki przy użyciu wywołania MQGMO_BROWSE_FIRST.

Kolejki w kolejności priorytetów

Pierwszym komunikatem w kolejce w tej sekwencji jest komunikat, który był w kolejce najdłuższy i ma najwyższy priorytet w czasie, gdy wywołanie MQOPEN zostało wydane.

Użyj komendy MQGMO_BROWSE_NEXT, aby odczytać komunikaty w kolejce.

Kursor przeglądania wskazuje na następny komunikat, pracując z priorytetu pierwszego komunikatu, który ma zostać ukończy z komunikatem o najniższym priorytecie. Powoduje ona przeglądy wszystkich komunikatów umieszczonych w kolejce w tym czasie, o ile są one równe lub mniejsze od priorytetu komunikatu identyfikowanego przez bieżący kursor przeglądania.

Każdy komunikat umieszczony w kolejce o wyższym priorytecie może być przeglądany tylko przez:

- Otwarcie kolejki w celu ponownego przeglądania, w którym to momencie zostanie utworzony nowy kursor przeglądania
- Korzystanie z opcji MQGMO_BROWSE_FIRST

Niezatwierdzone komunikaty

Niezatwierdzony komunikat nigdy nie jest widoczny dla przeglądania; kursor przeglądania przeskoczy obok niego.

Komunikaty w jednostce pracy nie mogą być przeglądane do momentu zatwierdzenia jednostki pracy. Komunikaty nie zmieniają swojej pozycji w kolejce po zatwierdzeniu, tak więc pominięte, niezatwierdzone komunikaty nie będą widoczne, nawet jeśli *są* zatwierdzane, chyba że używana jest opcja MQGMO_BROWSE_FIRST i ponownie działają mimo kolejki.

Zmiana kolejności kolejek

Jeśli sekwencja dostarczania komunikatów została zmieniona z priorytetu na FIFO, a w kolejce znajdują się komunikaty, to kolejność komunikatów, które są już umieszczone w kolejce, nie jest zmieniana. Komunikaty dodane do kolejki później, przyjmują domyślny priorytet kolejki.

Korzystanie z indeksu kolejki

Podczas przeglądania indeksowanej kolejki, która zawiera tylko komunikaty o pojedynczym priorytecie (trwałe lub nietrwałe), menedżer kolejek używa indeksu do przeglądania, gdy używane są pewne formy przeglądania.

Uwaga: Obsługiwane tylko w systemie IBM MQ for z/OS.

Jeśli indeksowana kolejka zawiera tylko komunikaty o pojedynczym priorytecie, używane są dowolne z poniższych form przeglądania:

1. Jeśli kolejka jest indeksowana przez MSGID, przeglądanie żądań, które przekaże identyfikator MSGID w strukturze MQMD, są przetwarzane przy użyciu indeksu w celu znalezienia komunikatu docelowego.
2. Jeśli kolejka jest indeksowana według identyfikatora CORRELID, przeglądanie żądań, które przekaże identyfikator CORRELID w strukturze MQMD, są przetwarzane przy użyciu indeksu w celu znalezienia komunikatu docelowego.
3. Jeśli kolejka jest indeksowana przez parametr GROUPID, przeglądanie żądań, które przekaże 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 jest indeksowana i zwracany jest komunikat, musi zostać znaleziony wpis indeksu dla komunikatu oraz informacje w niej używane do aktualizowania kursora przeglądania. Jeśli używany jest szeroki wybór wartości indeksu, nie spowoduje to dodatkowego przetwarzania dodatkowego żądania przeglądania.

Przeglądanie komunikatów, gdy długość komunikatu jest nieznana

Aby przejrzeć komunikat, jeśli nie znasz wielkości komunikatu i nie chcesz używać pól *MsgId*, *CorrelId*lub *GroupId* w celu znalezienia komunikatu, można użyć opcji MQGMO_BROWSE_MSG_UNDER_CURSOR:

1. Wydaj komendę MQGET z:

- Albo opcja MQGMO_BROWSE_FIRST, albo MQGMO_BROWSE_NEXT
- Opcja MQGMO_ACCEPT_TRUNCATED_MSG
- Długość buforu zerowego

Uwaga: Jeśli istnieje prawdopodobieństwo, że inny program będzie miał ten sam komunikat, należy rozważyć użycie opcji MQGMO_LOCK. Wartość MQRC_TRUNCATED_MSG_ACCEPTED powinna zostać zwrócona.

2. Użyj zwróconego *DataLength* , aby przydzielić potrzebną pamięć masową.
3. Wydaj komendę MQGET z opcją MQGMO_BROWSE_MSG_UNDER_CURSOR.

Komunikat wskazywał, że jest on ostatnim, który został pobrany. Kursor przeglądania nie zostanie przeniesiony. Można wybrać albo zablokować komunikat za pomocą opcji MQGMO_LOCK, albo odblokować zablokowany komunikat za pomocą opcji MQGMO_UNLOCK.

Wywołanie nie powiedzie się, jeśli żadna operacja MQGET z opcjami MQGMO_BROWSE_FIRST lub MQGMO_BROWSE_NEXT nie została pomyślnie wykonana, ponieważ kolejka została otwarta.

Usuwanie wiadomości, które zostały przejrzane
Można usunąć z kolejki komunikat, który został już przejrzany, pod warunkiem, że otwarto kolejkę do usuwania komunikatów, jak również do przeglądania. (Należy określić jedną z opcji MQOO_INPUT_ *, a także opcję MQOO_BROWSE w wywołaniu MQOPEN).

Aby usunąć komunikat, wywołaj komendę MQGET ponownie, ale w polu *Options* struktury MQGMO podaj wartość MQGMO_MSG_UNDER_CURSOR.W tym przypadku wywołanie MQGET zignoruje pola *MsgId*, *CorrelId*i *GroupId* struktury MQMD.

W czasie między krokami przeglądania i usuwania, inny program mógł usunąć komunikaty z kolejki, w tym komunikat pod kursorem przeglądania. W tym przypadku wywołanie MQGET zwraca kod przyczyny, aby stwierdzić, że komunikat nie jest dostępny.

Przeglądanie komunikatów w porządku logicznym

"Uporządkowanie logiczne i fizyczne" na stronie 866 wyjaśnia różnicę między logicznym i fizycznym porządkiem komunikatów w kolejce. To rozróżnienie jest szczególnie ważne podczas przeglądania kolejki, ponieważ w ogólnym przypadku komunikaty nie są usuwane, a operacje przeglądania nie muszą zaczynać się od początku kolejki.

Jeśli aplikacja przegląda różne komunikaty jednej grupy (przy użyciu porządku logicznego), ważne jest, aby po zamówieniu logicznym osiągnąć początek następnej grupy, ponieważ ostatni komunikat jednej grupy może wystąpić fizycznie *po* pierwszym komunikacie następnej grupy. Opcja MQGMO_LOGICAL_ORDER zapewnia, że podczas skanowania kolejki następuje porządek logiczny.

Użyj funkcji MQGMO_ALL_MSGS_AVAILABLE (lub MQGMO_ALL_SEGMENTS_AVAILABLE), aby zachować ostrożność podczas przeglądania operacji. Należy rozważyć przypadek komunikatów logicznych za pomocą komendy MQGMO_ALL_MSGS_AVAILABLE. Wynika to z tego, że komunikat logiczny jest dostępny tylko wtedy, gdy wszystkie pozostałe komunikaty w grupie są również obecne. Jeśli tak nie jest, komunikat zostanie przekazany. Może to oznaczać, że gdy brakujące komunikaty nadejdą później, nie są one zauważane przez operację przeglądania.

Na przykład, jeśli obecne są 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 funkcja przeglądania jest wydawana za pomocą MQGMO_ALL_MSGS_AVAILABLE, zwracany jest pierwszy komunikat logiczny grupy 456, pozostawiając kursor przeglądania w tym komunikacie logicznym. Jeśli zostanie wyświetlony 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-następna jest wydawana, nie jest zauważona, że grupa 123 jest teraz zakończona, ponieważ pierwszym komunikatem tej grupy jest *przed* kursorem przeglądania.

W niektórych przypadkach (na przykład, jeśli komunikaty są pobierane w sposób destruktywny, gdy grupa jest obecna w całości), można użyć komendy MQGMO_ALL_MSGS_AVAILABLE razem z MQGMO_BROWSE_FIRST. W przeciwnym razie należy powtórzyć skanowanie przeglądania w celu uwzględnienia nowo pojawiających się komunikatów, które zostały pominięte. Po prostu wydanie komendy MQGMO_WAIT razem z MQGMO_BROWSE_NEXT i MQGMO_ALL_MSGS_AVAILABLE nie uwzględnia ich. (To również dzieje się z komunikatami o wyższym priorytecie, które mogą nadejście po zakończeniu skanowania komunikatów).

W następnych sekcjach znajdują się przykłady przeglądania, które dotyczą nieposegmentowanych komunikatów, a 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 pogrupowanych komunikatów aplikacja nie chce uruchamiać przetwarzania żadnej grupy, dopóki wszystkie komunikaty w jej obrębie nie zostaną odebrane. Wartość MQGMO_ALL_ALL_MSGS_AVAILABLE jest więc określona dla pierwszego komunikatu w grupie; dla kolejnych komunikatów w grupie opcja ta jest zbędna.

W tym przykładzie użyto komendy MQGMO_WAIT. Jednak mimo że oczekiwanie może być spełnione, jeśli pojawi się nowa grupa z przyczyn w programie “Przeglądanie komunikatów w porządku logicznym” na stronie 897, nie jest ona spełniona, jeśli kursor przeglądania przeszedł już pierwszy komunikat logiczny w grupie, a pozostałe komunikaty są teraz przesyłane. Mimo to oczekiwanie na odpowiedni okres zapewnia, że aplikacja nie będzie stale zapętlała podczas oczekiwania na nowe komunikaty lub segmenty.

Parametr MQGMO_LOGICAL_ORDER jest używany przez cały czas w celu zapewnienia, że skanowanie jest w porządku logicznym. Kontrastuje to z destrukcyjnym przykładem MQGET, w którym ponieważ każda grupa jest usuwana, MQGMO_LOGICAL_ORDER nie jest używany w przypadku wyszukiwania pierwszego (lub tylko) komunikatu w grupie.

Zakłada się, że bufor aplikacji jest zawsze na tyle duży, aby pomieścić cały komunikat, niezależnie od tego, czy komunikat został podzielony na segmenty. W związku z tym 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, dopóki nie zostanie zwrócona wartość MQRC_NO_MSG_AVAILABLE.

Przeglądanie i pobieranie destrukcyjnie
W tym przykładzie aplikacja przegląda wszystkie komunikaty logiczne w grupie przed podjęciem decyzji o tym, czy ta grupa ma być odtwarzająca w sposób destruktywny.

Pierwsza część tego przykładu jest podobna do poprzedniej. Jednak w tym przypadku, po przeglądowaniu całej grupy, decydujemy się na cofanie i wydobycie jej destrukcyjnie.

Ponieważ każda grupa jest usuwana w tym przykładzie, parametr MQGMO_LOGICAL_ORDER nie jest używany w przypadku wyszukiwania pierwszego lub jedynego komunikatu w grupie.

Poniżej przedstawiono przykład przeglądania, a następnie odtwarzania destrukcyjnego:

```
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 powtarzającej się dostawy przejrzanych wiadomości
Korzystając z niektórych opcji otwierania i pobierania wiadomości, można oznaczyć wiadomości jako przeglądane, tak aby nie były one ponownie pobierane przez bieżące lub inne aplikacje współpracujące. Komunikaty mogą być nieoznaczone jawnie lub automatycznie, aby były ponownie dostępne do przeglądania.

Jeśli komunikaty są przeglądane w kolejce, można je pobrać w innej kolejności, w kolejności, w jakiej zostały pobrane, jeśli zostały one destrukcyjne. W szczególności można przeglądać ten sam komunikat wiele razy, co nie jest możliwe, jeśli jest on usuwany z kolejki. Aby tego uniknąć, można *oznaczyć* komunikaty w miarę ich przeglądania, a także unikać pobierania oznaczonych wiadomości. Czasami jest to określane jako *przeglądanie z zaznaczonym znakiem*. Aby oznaczyć przeglądane komunikaty, użyj opcji get message MQGMO_MARK_BROWSE_HANDLE, a w celu pobrania tylko komunikatów, które nie są oznaczone, użyj komendy MQGMO_UNMARKED_BROWSE_MSG. W przypadku użycia kombinacji opcji MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG i MQGMO_MARK_BROWSE_HANDLE i powtórzonych operacji MQGET, każdy komunikat w kolejce zostanie pobrany z kolei. Zapobiega to powtarzającej się dostarczania komunikatów, nawet jeśli używana jest komenda MQGMO_BROWSE_FIRST w celu zapewnienia, że komunikaty nie zostaną pominięte. 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, może otworzyć kolejkę z opcjami MQOO_CO_OP i MQOO_BROWSE. Uchwyt obiektu zwracany przez każdą operację MQOPEN uważa się za część współpracującej grupy. Dowolny komunikat zwrócony przez wywołanie MQGET, określający opcję MQGMO_MARK_BROWSE_CO_OP, jest uznawana za oznaczoną dla tego współpracującego zestawu uchwytów.

Jeśli komunikat został oznaczony przez pewien czas, może on być automatycznie nieoznaczony przez menedżer kolejek i udostępniony do ponownego przeglądania. Atrybut MsgMarkmenedżera kolejekBrowseInterval określa czas (w milisekundach), przez który komunikat ma pozostać oznaczony jako odpowiedni dla współpracującego zestawu uchwytów. Wartość MsgMarkBrowseInterval równa -1 oznacza, że komunikaty nigdy nie są automatycznie nieoznaczone.

Gdy pojedynczy proces lub zestaw kooperatywnych procesów zaznacza komunikaty zatrzymują się, wszystkie oznaczone komunikaty stają się nieoznaczone.

Przykłady przeglądania kooperatywnego

Istnieje możliwość uruchomienia wielu kopii aplikacji programu rozsyłającego 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 programy rozsyłające współpracują ze sobą i będą mieć świadomość, że są one oznaczone jako komunikaty. Następnie każdy program rozsyłający wykonuje powtarzające się wywołania MQGET, określając opcje MQGMO_BROWSE_FIRST, MQGMO_UNMARKED_BROWSE_MSG i MQGMO_MARK_BROWSE_CO_OP (można użyć pojedynczej stałej MQGMO_BROWSE_CO_OP, aby reprezentować tę kombinację opcji). Każda aplikacja programu rozsyłającego pobiera następnie tylko te komunikaty, które nie zostały jeszcze oznaczone przez inne współpracujące programy rozsyłające. Program rozsyłający inicjuje konsumenta i przekazuje element MsgToken zwracany przez komendę MQGET do konsumenta, co w sposób destruktywny pobiera komunikat z kolejki. Jeśli konsument wycofał operację MQGET komunikatu, komunikat jest dostępny dla jednej z przeglądarek w celu ponownego wysłania, ponieważ nie jest już oznaczony. Jeśli konsument nie wykonuje operacji MQGET w komunikacie, po upływie wartości MsgMarkBrowseInterval , menedżer kolejek odznaczy komunikat dla współpracującego zestawu uchwytów i może zostać ponownie rozesłany.

Zamiast wielu kopii tej samej aplikacji programu rozsyłającego, użytkownik może mieć kilka różnych aplikacji programu rozsyłającego, które przeglądają kolejkę, a każda z nich jest odpowiednia do przetwarzania podzbioru komunikatów w kolejce. W każdym programie rozsyłającym otwórz kolejkę za pomocą komendy MQOO_CO_OP. Oznacza to, że programy rozsyłające współpracują ze sobą i będą mieć świadomość, że są one oznaczone jako komunikaty.

- Jeśli kolejność przetwarzania komunikatów dla pojedynczego przekaźnika jest ważna, 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 przejrzany komunikat jest odpowiedni dla tego programu rozsyłającego do przetwarzania, to wywoła wywołanie MQGET z określeniem wartości MQMO_MATCH_MSG_TOKEN, MQGMO_MARK_BROWSE_CO_OP i MsgToken zwróconego przez poprzednie wywołanie MQGET. Jeśli wywołanie powiedzie się, program rozsyłający inicjuje konsumenta, przekazując do niego element MsgToken .
- Jeśli kolejność przetwarzania komunikatów nie jest istotna, a oczekuje się, że program rozsyłający będzie przetwarzać większość komunikatów, które napotyka, 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 można przetworzyć, usuń ten komunikat, wywołując komendę MQGET z opcją MQMO_MATCH_MSG_TOKEN, MQGMO_UNMARK_BROWSE_CO_OP i MsgToken zwróconej poprzednio.

Niektóre przypadki, w których wywołanie MQGET nie powiodło się

Jeśli niektóre atrybuty kolejki zostaną zmienione za pomocą opcji FORCE w komendzie między wywołaniem wywołania MQOPEN i wywołaniem MQGET, wywołanie MQGET nie powiedzie się i zwróci kod przyczyny MQRC_OBJECT_CHANGED.

Menedżer kolejek oznacza, że uchwyt obiektu nie jest już poprawny. Dzieje się tak również wtedy, gdy zmiany będą miały zastosowanie do każdej kolejki, do której nazwa kolejki jest tłumaczona. Atrybuty, które mają wpływ na uchwyt w ten sposób, są wymienione w opisie wywołania MQOPEN w produkcie MQOPEN. Jeśli wywołanie zwróci kod przyczyny MQRC_OBJECT_CHANGED, zamknij kolejkę, ponownie go otwórz, a następnie spróbuj ponownie uzyskać komunikat.

Jeśli operacje pobierania są zablokowane dla kolejki, z której podjęto próbę pobrania komunikatów (lub dowolnej kolejki, do której nazwa kolejki jest tłumaczona), wywołanie MQGET nie powiedzie się i zwróci kod przyczyny MQRC_GET_INHIBITED. Zdarza się to nawet w przypadku korzystania z wywołania MQGET w celu przeglądania. Może być możliwe pomyślne pobranie komunikatu w przypadku podjęcia próby wywołania MQGET w późniejszym czasie, jeśli projekt aplikacji jest taki, że inne programy regularnie zmieniają atrybuty kolejek.

Jeśli usunięto kolejkę dynamiczną (tymczasową lub trwałą), wywołania MQGET z użyciem wcześniej uzyskanego uchwytu obiektu nie powiodą się i zwróć kod przyczyny MQRC_Q_DELETED.

Zapisywanie aplikacji publikowania/subskrypcji

Rozpocznij pisanie aplikacji publikowania/subskrypcji IBM MQ .

Przegląd pojęć związanych z publikowaniem/subskrybowaniem zawiera sekcja Przesyłanie komunikatów w trybie publikowania/subskrypcji.

Informacje na temat pisania różnych typów aplikacji publikowania/subskrypcji można znaleźć w następujących tematach:

- “Zapisywanie aplikacji publikatorów” na stronie 902
- “Pisanie aplikacji subskrybentów” na stronie 908
- “Cykle życia publikowania/subskrybowania” na stronie 926
- “Właściwości komunikatu publikowania/subskrypcji” na stronie 931
- “Porządkowanie komunikatów” na stronie 933
- “Przechwytywanie publikacji” na stronie 933
- “Opcje publikowania” na stronie 941
- “Opcje subskrypcji” na stronie 941

Pojęcia pokrewne

“Pojęcia związane z projektowaniem aplikacji” na stronie 7
Do pisania aplikacji IBM MQ można użyć wyboru języków proceduralnych lub obiektowych. Przed rozpoczęciem projektowania i pisania aplikacji produktu IBM MQ należy zapoznać się z podstawowymi pojęciami dotyczącymi produktu IBM MQ .

“Tworzenie aplikacji dla składnika IBM MQ” na stronie 5
Istnieje możliwość tworzenia aplikacji w celu wysyłania i odbierania komunikatów oraz do zarządzania menedżerami kolejek i powiązanymi zasobami. Produkt IBM MQ obsługuje aplikacje napisane w wielu różnych językach i w różnych ramach.

“Uwagi dotyczące projektowania aplikacji produktu IBM MQ” na stronie 50
Po zdecydowaniu, w jaki sposób aplikacje mogą korzystać z platform i środowisk, które są dostępne dla użytkownika, należy zdecydować, w jaki sposób korzystać z funkcji oferowanych przez produkt IBM MQ.

“Pisanie aplikacji proceduralnej w celu kolejkowania” na stronie 810
Ta sekcja zawiera informacje na temat pisania aplikacji kolejkowania, łączenia się i rozłączania z menedżerem kolejek, publikowania/subskrypcji oraz obiektów otwierających i zamykających.

“Pisanie aplikacji proceduralnych klienta” na stronie 1008
Co należy wiedzieć, aby pisać aplikacje klienckie w systemie IBM MQ , korzystając z języka proceduralnego.

“Budowanie aplikacji proceduralnej” na stronie 1099
Aplikację IBM MQ można napisać w jednym z kilku języków proceduralnych, a następnie uruchomić aplikację na kilku różnych platformach.

“Obsługa proceduralnych błędów programu” na stronie 1145
Te informacje wyjaśniają błędy związane z wywołaniami MQI aplikacji, gdy wywołuje połączenie, lub gdy jego komunikat jest dostarczany do miejsca docelowego.

Zadania pokrewne

“Korzystanie z przykładowych programów proceduralnych produktu IBM MQ” na stronie 1165

Te przykładowe programy są zapisywane w językach proceduralnych i demonstrują typowe zastosowania interfejsu kolejki komunikatów (Message Queue Interface-MQI). Programy IBM MQ na różnych platformach.

Zapisywanie aplikacji publikatorów

Rozpoczęcie pracy z pisaniem aplikacji publikatorów poprzez zapoznanie się z dwoma przykładami. Pierwsza z nich jest możliwie najdokładniej modelowana w punkcie, w którym aplikacja wskazuje na umieszczanie komunikatów w kolejce, a druga demonstruje dynamicznie tworzenie tematów-bardziej powszechny wzorzec dla aplikacji publikatorów.

Zapisywanie prostej aplikacji publikatora produktu IBM MQ jest tak samo, jak zapisywanie punktu IBM MQ do punktu aplikacji, w którym komunikaty są umieszczane w kolejce (Tabela 125 na stronie 902). Różnica polega na tym, że komunikaty MQPUT są wysyłane do tematu, a nie do kolejki.

Tabela 125. Point to point versus publish/subscribe IBM MQ program pattern.

Krok	**Punkt z punktem MQ Call**	**Publikuj wywołanie MQ**
Połącz z menedżerem kolejek	MQCONN	MQCONN
Otwarta kolejka	MQOPEN	
Otwórz wątek		MQOPEN
Umieść komunikat (y)	MQPUT	MQPUT
Zamknij temat		MQCLOSE
Zamknij kolejkę	MQCLOSE	
Rozłącz z menedżerem kolejek	MQDISC	MQDISC

Aby zrobić ten konkretny, istnieją dwa przykłady zastosowań do publikowania cen akcji. W pierwszym przykładzie (“Przykład 1: publikator w stałym temacie” na stronie 902), który jest ściśle modelowany przy umieszczanie komunikatów w kolejce, administrator tworzy definicję tematu w podobny sposób, aby utworzyć kolejkę. Programmer kodów MQPUT służy do zapisywania komunikatów w temacie, a nie do zapisywania ich w kolejce. W drugim przykładzie (“Przykład 2: publikator w temacie o zmiennej” na stronie 905) wzorzec interakcji programu z IBM MQ jest podobny. Różnica polega na tym, że programista udostępnia temat, do którego jest zapisywany komunikat, a nie przez administratora. W praktyce oznacza to zwykle, że łańcuch tematu jest zdefiniowany przez treść lub jest udostępniany przez inne źródło, takie jak dane wprowadzane przez użytkownika przez przeglądarkę.

Pojęcia pokrewne

“Pisanie aplikacji subskrybentów” na stronie 908
Rozpoczynanie pracy z pisaniem aplikacji subskrybentów przez zapoznanie się z trzema przykładami: aplikacja IBM MQ konsumująca komunikaty z kolejki, aplikacja tworująca subskrypcję i nie wymaga znajomości kolejkowania, a w końcu przykład, który używa zarówno kolejkowania, jak i subskrypcji.

Odsyłacze pokrewne

ZDEFINIUJ TEMAT
WYŚWIETL TEMAT
WYŚWIETL STATUS TPSTATUS

Przykład 1: publikator w stałym temacie

Program IBM MQ w celu zilustrować publikowanie do tematu zdefiniowanego administracyjnie.

Uwaga: Kompaktowy styl kodowania jest przeznaczony do czytelności, a nie do użytku produkcyjnego.

Patrz dane wyjściowe w programie Rysunek 70 na stronie 903 .

```
#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 69. Prosty publikator produktu 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 70. Przykładowe dane wyjściowe z pierwszego przykładu publikatora

Następujące 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 dla programu.

MQCHAR resTopicStr[151];
Produkt `resTopicStr` jest wskazyany przez produkt `td.ResObjectString.VSPtr` i jest używany przez produkt `MQOPEN` do zwracania przetłumaczanego łańcucha tematu. Ustaw długość `resTopicStr` o jedną większą niż długość, która została przekazana w programie `td.ResObjectString.VSBufSize` , aby zapewnić miejsce na zerowe zakończenie.

memset (resTopicStr, 0, sizeof(resTopicStr));
Zainicjuj parametr `resTopicStr` w celu usunięcia wartości NULL, aby upewnić się, że rozstrzygnięty łańcuch tematu zwrócony w polu `MQCHARV` ma wartość NULL.

td.ObjectType = MQOT_TOPIC
Istnieje nowy typ obiektu dla publikowania/subskrypcji: *obiekt tematu*.

td.Version = MQOD_VERSION_4;
Aby korzystać z nowego typu obiektu, należy użyć co najmniej *wersji 4* deskryptora obiektu.

strncpy(td.ObjectName, topicName, MQ_OBJECT_NAME_LENGTH);
`topicName` jest nazwą obiektu tematu, czasami nazywanych obiektem tematu administracyjnego. W tym przykładzie należy wcześniej utworzyć obiekt tematu przy użyciu programu IBM MQ Explorer lub tej komendy MQSC,

```
DEFINE TOPIC(IBMSTOCKPRICE) TOPICSTR(NYSE/IBM/PRICE) REPLACE;
```

td.ResObjectString.VSPtr = resTopicStr;
Rozstrzygnięty łańcuch tematu jest odbity w finalnym `printf` w programie. Skonfiguruj strukturę `MQCHARV ResObjectString` dla programu IBM MQ , aby przywrócić rozstrzygnięty łańcuch z powrotem do programu.

MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
Otwórz temat dla danych wyjściowych, tak jak otwieranie kolejki dla danych wyjściowych.

pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_RETAIN;
Użytkownik chce, aby nowi subskrybenci mieli możliwość otrzymywania publikacji, a także poprzez podanie wartości `MQPMO_RETAIN` w publikatorze po uruchomieniu subskrybenta, który otrzymuje najnowszą publikację, opublikowaną przed uruchomieniem subskrybenta, jako pierwszą zgodną z nią publikację. Alternatywą jest udostępnienie subskrybentom publikacji opublikowanych dopiero po uruchomieniu subskrybenta. Dodatkowo subskrybent ma możliwość odrzucenia zachowanej publikacji przez określenie wartości `MQSO_NEW_PUBLICATIONS_ONLY` w subskrypcji.

MQPUT(Hconn, Hobj, &md, &pmo, (MQLONG)strlen(publication)+1, publication, &CompCode, &Reason);
Dodaj wartość 1 do długości łańcucha przekazanego do programu `MQPUT` , aby przekazać znak zakończenia o wartości NULL do IBM MQ jako część buforu komunikatów.

Co przedstawia pierwszy przykład? Przykład naśladuje możliwie najdokładniej wypróbowany i testowany tradycyjny wzorzec do pisania punktowego programów IBM MQ . Istotną cechą wzorca programowania IBM MQ jest to, że programista nie jest zaniepokojony, gdy wysyłane są komunikaty. Zadaniem programisty jest połączenie się z menedżerem kolejek i przekazanie mu komunikatów, które mają być dystrybuowane do odbiorców. W paradygmach punkt-punkt programista otwiera kolejkę (prawdopodobnie kolejkę aliasową), którą skonfigurował administrator. Kolejka aliasowa kieruje komunikaty do kolejki docelowej albo w lokalnym menedżerze kolejek, albo do menedżera kolejek zdalnych. Podczas gdy komunikaty oczekują na dostarczenie, są one przechowywane w kolejkach między źródłem a miejscem docelowym.

W przypadku wzorca publikowania/subskrypcji, zamiast otwierania kolejki programista otwiera temat. W naszym przykładzie temat jest powiązany z łańcuchem tematu przez administratora. Menedżer kolejek przekazuje publikację, korzystając z 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 już żadnych subskrybentów. Zachowana

publikacja jest dostępna do przekazania do przyszłych subskrybentów. Aplikacja publikująca nie odgrywa żadnej części w wyborze lub kierowaniu publikacji do miejsca docelowego; jej zadaniem jest tworzenie i umieszczanie publikacji na tematy zdefiniowane przez administratora.

Ten przykład stałego tematu jest nietypowy dla wielu aplikacji publikowania/subskrypcji: jest on statyczny. Wymaga on od administratora zdefiniowania łańcuchów tematów i zmiany tematów, które są publikowane. Często aplikacje publikowania/subskrybowania muszą znać niektóre lub wszystkie drzewa tematów. Być może tematy często zmieniają się lub być może mimo że tematy nie zmieniają się zbyt wiele, liczba kombinacji tematów jest duża i zbyt uciążliwe dla administratora, aby zdefiniować węzeł tematu dla każdego łańcucha tematu, który może wymagać opublikowania. Być może łańcuchy tematów nie są znane przed publikacją. Aplikacja publikująca może używać informacji z treści publikacji w celu określenia łańcucha tematu lub może zawierać informacje o łańcuchach tematów do opublikowania w innym źródle, takim jak dane wprowadzane przez użytkownika z przeglądarki. Aby uzyskać bardziej dynamiczne style publikowania, w następnym przykładzie przedstawiono sposób dynamicznego tworzenia tematów w ramach aplikacji publikatora.

Wątki publikatorów i subskrybentów razem. Projektowanie reguł lub architektury na potrzeby nazewnictwa tematów oraz organizowanie ich w drzewach tematów jest ważnym krokiem w tworzeniu rozwiązania publikowania/subskrypcji. Należy uważnie przyjrzeć się, w jakim stopniu organizacja drzewa tematów łączy programy publikatorów i subskrybentów, a następnie wiąże je z treścią drzewa tematów. Zadaj sobie pytanie, czy zmiany w drzewie tematów wpływają na aplikacje publikatorów i subskrybentów oraz jak można zminimalizować efekt. Wbudowana architektura modelu publikowania/subskrypcji produktu IBM MQ jest pojęciem obiektu tematu administracyjnego, który udostępnia część główną lub poddrzewo główne tematu. Obiekt tematu umożliwia zdefiniowanie głównej części drzewa tematów w sposób administracyjny, który upraszcza programowanie aplikacji i operacje, a co za tym samym zwiększa łatwość konserwacji. Na przykład w przypadku wdrażania wielu aplikacji publikowania/subskrybowania, które mają izolowane drzewa tematów, a następnie administracyjnie definiując główną część drzewa tematów, można zagwarantować odseparowanie drzew tematów, nawet jeśli nie ma spójności konwencji nazewnictwa tematów przyjętych przez różne aplikacje.

W praktyce aplikacje wydawcy obejmują spektrum od wyłącznie korzystania z stałych tematów, jak w tym przykładzie, i tematów zmiennych, jak w następnym. Program “Przykład 2: publikator w temacie o zmiennej” na stronie 905 demonstruje również połączenie użycia tematów i łańcuchów tematów.

Pojęcia pokrewne

“Przykład 2: publikator w temacie o zmiennej” na stronie 905
Program WebSphere MQ w celu zilustrować publikowanie w programowo zdefiniowanym temacie.

“Pisanie aplikacji subskrybentów” na stronie 908
Rozpoczynanie pracy z pisaniem aplikacji subskrybentów przez zapoznanie się z trzema przykładami: aplikacja IBM MQ konsumująca komunikaty z kolejki, aplikacja tworująca subskrypcję i nie wymaga znajomości kolejkowania, a w końcu przykład, który używa zarówno kolejkowania, jak i subskrypcji.

Przykład 2: publikator w temacie o zmiennej
Program WebSphere MQ w celu zilustrować publikowanie w programowo zdefiniowanym temacie.

Uwaga: Kompaktowy styl kodowania jest przeznaczony do czytelności, a nie do użytku produkcyjnego.

Patrz dane wyjściowe w programie Rysunek 72 na stronie 906.

```
#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 71. Prosty publikator produktu IBM MQ do 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 72. Przykładowe dane wyjściowe z drugiego przykładu publikatora

Jest kilka punktów, które warto zwrócić uwagę na ten przykład.

char topicNameDefault[] = "STOCKS";
Domyślna nazwa tematu STOCKS definiuje część łańcucha tematu. Tę nazwę tematu można przesłonić, udostępniając go jako pierwszy argument w programie lub eliminując użycie nazwy tematu przez podanie / jako pierwszego parametru.

char topicString[101] = "IBM/PRICE";
IBM/PRICE jest domyślnym łańcuchem tematu. Ten łańcuch tematu można przesłonić, udostępniając go jako drugi argument programu.

Menedżer kolejek łączy łańcuch tematu udostępniany przez obiekt tematu STOCKS "NYSE"z łańcuchem tematu udostępnionym przez program "IBM/PRICE" i wstawia element "/" między dwoma łańcuchami tematów. Wynikiem jest rozstrzygnięty łańcuch tematu "NYSE/IBM/PRICE". Wynikowy łańcuch tematu jest taki sam, jak ten zdefiniowany w obiekcie tematu IBMSTOCKPRICE i ma on dokładnie taki sam efekt.

Obiekt tematu administracyjnego powiązany z rozstrzygniętym łańcuchem tematu nie musi być tym samym obiektem tematu, który został przekazany do produktu MQOPEN przez publikatora. Produkt IBM MQ używa drzewa niejawnego w rozstrzygniętym łańcuchu tematu, aby określić, który obiekt tematu administracyjnego definiuje atrybuty powiązane z publikacją.

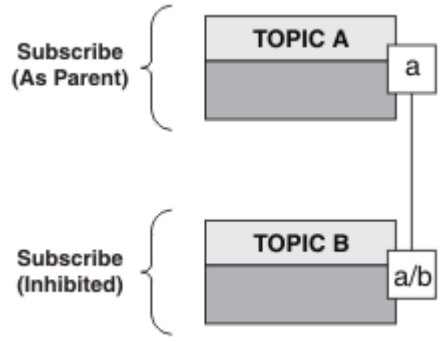
Załóżmy, że istnieją dwa obiekty tematów A i B, a A definiuje temat "a", a B definiuje temat "a/b" (Rysunek 73 na stronie 907). Jeśli program publikujący odwołuje się do obiektu tematu A i zawiera łańcuch tematu "b", rozstrzygając temat w łańcuchu tematu "a/b", opublikowanie dziedziczy jego właściwości 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 udostępnionym topicName. "/" jest niepoprawna jako nazwa tematu. Oznacza to, że nie ma żadnej nazwy tematu, a łańcuch tematu jest w całości udostępniany przez program. Dane wyjściowe w programie Rysunek 72 na stronie 906 przedstawiają cały łańcuch tematu, który jest dostępny dynamicznie przez program.

strncpy(td.ObjectName, topicName, MQ_OBJECT_NAME_LENGTH);
W przypadku domyślnego elementu pracy należy wcześniej utworzyć opcjonalną topicName za pomocą programu IBM MQ Explorer lub za pomocą tej komendy MQSC:

```
DEFINE TOPIC(STOCKS) TOPICSTR(NYSE) REPLACE;
```

td.ObjectString.VSPtr = topicString;
Łańcuch tematu to pole MQCHARV w deskryptorze tematu.



Rysunek 73. Powiązania obiektów tematu

Co ilustruje drugi przykład? Chociaż kod jest bardzo podobny do pierwszego przykładu-skutecznie istnieją tylko dwie linie różnicy-wynik jest zdecydowanie odmiennym programem do pierwszego. Programista kontroluje miejsca docelowe, do których wysyłane są publikacje. W połączeniu z minimalnym wejściem

administratora używanym do projektowania aplikacji subskrybentów, żadne tematy ani kolejki nie muszą być wstępnie zdefiniowane w celu kierowania publikacji z publikatorów do subskrybentów.

W paradygmach przesyłania komunikatów w trybie punkt z punktem należy zdefiniować kolejki, zanim komunikaty będą mogły przepływać. W przypadku publikowania/subskrypcji nie są one używane, chociaż produkt IBM MQ implementuje interfejs publikowania/subskrypcji przy użyciu bazowego systemu kolejkowania. Korzyści wynikające z gwarantowanego dostarczania, transakcyjności i luźnego połączenia powiązanego z przesyłaniem komunikatów i kolejkowaniem są dziedziczone przez aplikacje publikowania/subskrypcji.

Projektant musi zdecydować, czy publikator i subskrybent mają mieć świadomość bazowego drzewa tematów, czy też nie, a także czy programy subskrybentów są świadome kolejkowania, czy nie. Następnie należy zbadać przykładowe aplikacje subskrybenta. Są one przeznaczone do użycia z przykładami publikatora, zwykle publikowaniem i subskrybowaniem produktu `NYSE/IBM/PRICE`.

Pojęcia pokrewne

“Przykład 1: publikator w stałym temacie” na stronie 902
Program IBM MQ w celu zilustrować publikowanie do tematu zdefiniowanego administracyjnie.

“Pisanie aplikacji subskrybentów” na stronie 908
Rozpoczynanie pracy z pisaniem aplikacji subskrybentów przez zapoznanie się z trzema przykładami: aplikacja IBM MQ konsumująca komunikaty z kolejki, aplikacja tworująca subskrypcję i nie wymaga znajomości kolejkowania, a w końcu przykład, który używa zarówno kolejkowania, jak i subskrypcji.

Pisanie aplikacji subskrybentów

Rozpoczynanie pracy z pisaniem aplikacji subskrybentów przez zapoznanie się z trzema przykładami: aplikacja IBM MQ konsumująca komunikaty z kolejki, aplikacja tworująca subskrypcję i nie wymaga znajomości kolejkowania, a w końcu przykład, który używa zarówno kolejkowania, jak i subskrypcji.

W programie Tabela 126 na stronie 908 wyświetlane są trzy style konsumenta lub subskrybenta, wraz z sekwencjami wywołań funkcji IBM MQ , które je charakteryzują.

1. Pierwszy styl, MQ Publication Consumer, jest taki sam, jak w przypadku programu MQ , który wykonuje tylko MQGET. Aplikacja nie ma wiedzy, że jest konsumowana w publikacjach-po prostu odczytuje wiadomości 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 subskrybentów. Aplikacja subskrybenta tworzy subskrypcję, a następnie uzyskuje publikacje. Zarządzanie kolejkami jest wykonywane przez menedżer kolejek.
3. W trzecim stylu aplikacja subskrybenta zdecyduje się na otwarcie i zamknięcie kolejki bazowej, która jest używana do publikacji, a także do emisji subskrypcji w celu wypełnienia kolejki publikacjami.

Jednym ze sposobów zrozumienia tych stylów jest zapoznanie się z przykładowymi programami C wymienionymi w programie Tabela 126 na stronie 908 dla każdego ze stylów. Przykłady zostały zaprojektowane w taki sposób, aby były uruchamiane w połączeniu z przykładem publikatora znalezionym w produkcie “Zapisywanie aplikacji publikatorów” na stronie 902.

Tabela 126. Point to point vs. subscribe IBM MQ program patterns.

Krok	**Konsument komunikatów MQ**	**“Przykład 1: konsument publikacji MQ” na stronie 909**	**“Przykład 2: zarządzany subskrybent produktu MQ” na stronie 911**	**“Przykład 3: Niezarządzany subskrybent MQ” na stronie 916**
Połącz z menedżerem kolejek	MQCONN	MQCONN	MQCONN	MQCONN
Otwarta kolejka	MQOPEN	MQOPEN		MQOPEN
Subskrybowanie			MQSUB	MQSUB

Tabela 126. Point to point vs. subscribe IBM MQ program patterns. (kontynuacja)				
Krok	**Konsument komunikatów MQ**	**"Przykład 1: konsument publikacji MQ" na stronie 909**	**"Przykład 2: zarządzany subskrybent produktu MQ" na stronie 911**	**"Przykład 3: Niezarządzany subskrybent MQ" na stronie 916**
Pobierz komunikat (y)	MQGET	MQGET	MQGET	MQGET
Zamknij kolejkę	MQCLOSE	MQCLOSE	(MQCLOSE)	MQCLOSE
Zamknij subskrypcję			MQCLOSE	MQCLOSE
Rozłącz z menedżerem kolejek	MQDISC	MQDISC	MQDISC	MQDISC

Użycie komendy MQCLOSE jest zawsze opcjonalne, aby zwolnić zasoby, przekazać opcje MQCLOSE lub tylko dla symetrii z opcją MQOPEN. Ponieważ nie jest konieczne określenie opcji MQCLOSE w sytuacji, gdy kolejka subskrypcji jest zamknięta w ramach subskrypcji zarządzanej MQ , a argument symetrii nie jest istotny, kolejka subskrypcji nie jest jawnie zamknięta w Przykład 2: Zarządzany subskrybent MQ.

Innym sposobem zrozumienia wzorców aplikacji publikowania/subskrypcji jest zbyt duże spojrzenie na interakcje między różnymi zaangażowanymi podmiotami. Diagramy linii Lifeline lub UML są dobrym sposobem na badanie interakcji. Trzy przykłady linii życia są opisane w podręczniku "Cykle życia publikowania/subskrybowania" na stronie 926.

Przykład 1: konsument publikacji MQ
Konsument publikowania produktu MQ jest konsumentem komunikatów produktu IBM MQ , który nie subskrybuje się do samych tematów.

Aby utworzyć kolejkę subskrypcji i publikacji dla tego przykładu, należy uruchomić następujące komendy lub zdefiniować 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 pamiętać, że przed utworzeniem subskrypcji konieczne jest zdefiniowanie obiektu tematu i kolejki używanej do odbierania publikacji.

Istnieje wiele cennych aspektów dla wzorca konsumenta publikacji produktu MQ :

1. Multiprocessing: dzielenie się z pracą lektur publikacji. Wszystkie publikacje znajdują się w pojedynczej kolejce powiązanej z tematem subskrypcji. Wiele konsumentów może otworzyć kolejkę za pomocą programu MQOO_INPUT_SHARED.
2. Centralnie zarządzane subskrypcje. Aplikacje nie konstruują własnych tematów subskrypcji ani subskrypcji. Administrator jest odpowiedzialny za to, gdzie są wysyłane publikacje.
3. Koncentracja subskrypcji: do jednej kolejki może zostać wysłana wiele różnych subskrypcji.
4. Trwałość subskrypcji: kolejka odbiera wszystkie publikacje, bez względu na to, czy konsumenci są aktywni.
5. Migracja i współistnienie: kod konsumenta działa równie dobrze w przypadku scenariusza "punkt z punktem" i "publikowania/subskrypcji".

Subskrypcja tworzy relację między łańcuchem tematu NYSE/IBM/PRICE a kolejką STOCKTICKER. Publikacje, w tym wszelkie aktualnie zachowane publikacje, są przekazywane do produktu STOCKTICKER od momentu utworzenia subskrypcji.

Można zarządzać lub zarządzać niezarządzaną subskrypcją administracyjną utworzoną przez administratora. Subskrypcja zarządzana staje się skuteczna od momentu jej utworzenia, podobnie jak subskrypcja niezarządzana. Nie wszystkie aspekty wzorca są dostępne dla subskrypcji zarządzanej. Więcej informacji znajduje się w sekcji “Przykład 3: Niezarządzany subskrybent MQ” na stronie 916

Uwaga: Kompaktowy styl kodowania jest przeznaczony do czytelności, a nie do użytku produkcyjnego.

Wyniki są wyświetlane w programie Rysunek 75 na stronie 911.

```
#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 74. Konsument publikowania produktu 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 75. Dane wyjściowe z konsumenta publikacji MQ

Istnieje kilka standardowych wskazówek do programowania w języku IBM MQ *C* , które mają być świadome:

memset(publication, 0, sizeof(publicationBuffer));
: Upewnij się, że na końcu komunikatu ma wartość NULL, aby ułatwić formatowanie przy użyciu produktu `printf`. Przykład publikatora zawiera końcowe wartości null w buforze komunikatów przekazanego do programu MQPUT przez dodanie wartości 1 do `strlen(publication)`. Ustawienie w buforach MQCHAR buforów o wartości NULL jest dobrym stylem programowania dla programów IBM MQ w języku C, które używają buforów do przechowywania łańcuchów, przy czym wartość NULL jest zgodna z tablicą znaków, która nie zapełnia całkowicie buforu.

MQGET(Hconn, Hobj, &md, &gmo, sizeof(publicationBuffer)-1, publication, &messlen, &CompCode, &Reason);
: Rezerwuj jedną wartość null na końcu buforu komunikatów, aby upewnić się, że zwrócony komunikat ma na końcu wartość NULL w przypadku, gdy `if (messlen == strlen(publication));` ma wartość true. Ta wskazówka stanowi uzupełnienie poprzedniego elementu i zapewnia, że w produkcie `publicationBuffer` jest co najmniej jedna wartość NULL, która nie jest nadpisywana przez zawartość produktu `publication`.

Pojęcia pokrewne

“Przykład 2: zarządzany subskrybent produktu MQ” na stronie 911
Zarządzany subskrybent MQ jest preferowanym wzorcem dla większości aplikacji subskrybentów. W tym przykładzie wymagana jest *nie* administracyjna definicja kolejek, tematów lub subskrypcji.

“Przykład 3: Niezarządzany subskrybent MQ” na stronie 916
Niezarządzany subskrybent jest ważną klasą aplikacji subskrybenta. Dzięki temu użytkownik łączy korzyści płynące z publikowania/subskrybowania za pomocą *sterowania* kolejkowania i korzystania z publikacji. W przykładzie przedstawiono różne sposoby łączenia subskrypcji i kolejek.

“Zapisywanie aplikacji publikatorów” na stronie 902
Rozpoczęcie pracy z pisaniem aplikacji publikatorów poprzez zapoznanie się z dwoma przykładami. Pierwsza z nich jest możliwie najdokładniej modelowana w punkcie, w którym aplikacja wskazuje na umieszczanie komunikatów w kolejce, a druga demonstruje dynamicznie tworzenie tematów-bardziej powszechny wzorzec dla aplikacji publikatorów.

Przykład 2: zarządzany subskrybent produktu MQ

Zarządzany subskrybent MQ jest preferowanym wzorcem dla większości aplikacji subskrybentów. W tym przykładzie wymagana jest *nie* administracyjna definicja kolejek, tematów lub subskrypcji.

Ten najprostszy rodzaj zarządzanego subskrybenta zwykle korzysta z subskrypcji *nietrwałej* . Ten przykład koncentruje się na nietrwałej subskrypcji. Subskrypcja trwa tylko tak długo, jak czas życia uchwytu subskrypcji z produktu MQSUB. Wszystkie publikacje, które są zgodne z łańcuchem tematu w czasie życia subskrypcji, są wysyłane do kolejki subskrypcji (i być może zachowaną publikacją, jeśli opcja MQSO_NEW_PUBLICATIONS_ONLY nie jest ustawiona lub została użyta domyślnie, wcześniejsza 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 żadnych błędów spowodowałaby przeżycie subskrybenta. Więcej informacji na temat różnych cykli życia powiązanych z subskrypcjami zarządzanymi, niezarządzanymi, trwałymi i nietrwałymi zawierają sekcje dotyczące tematów pokrewnych.

Subskrypcje trwałe są często powiązane z publikacjami trwałymi i subskrypcjami nietrwałymi z publikacjami nietrwałymi, ale nie ma potrzeby relacji między trwałością subskrypcji a trwałością publikacji. Możliwe są wszystkie cztery kombinacje trwałości i trwałości.

W przypadku rozpatrywanych nietrwałych obserwacji zarządzanych menedżer kolejek tworzy kolejkę subskrypcji, która jest czyszczona i usuwana po zamknięciu kolejki. Publikacje są usuwane z kolejki, gdy subskrypcja nietrwała jest zamknięta.

Cennymi aspektami zarządzanego nietrwałego wzorca, które są przykładowo następujące:

1. Subskrypcja na żądanie: łańcuch tematu subskrypcji jest dynamiczny. Jest ona udostępniana przez aplikację po jej uruchomieniu.
2. Kolejka samozarządzająca: kolejka subskrypcji jest samodefiniująca się i zarządzająca.
3. Cykl życia subskrypcji samozarządzającej: *nietrwałe* subskrypcje istnieją tylko na czas trwania aplikacji subskrybenta.
   - Jeśli zostanie zdefiniowana *trwała* subskrypcja zarządzana, to spowoduje to, że będzie ona przechowywana w trwałej kolejce subskrypcji, a publikacje będą przechowywane na niej bez aktywności żadnych programów subskrybentów. Menedżer kolejek usuwa kolejkę (i usuwa z niej wszystkie niepobrane publikacje) tylko po tym, jak aplikacja lub administrator wybrała usunięcie subskrypcji. Subskrypcję można usunąć za pomocą komendy administracyjnej lub zamykając subskrypcję za pomocą opcji `MQCO_REMOVE_SUB` .
   - Należy rozważyć ustawienie `SubExpiry` dla trwałych subskrypcji, tak aby publikacje przestały być wysyłane do kolejki, a subskrybent mógł korzystać z pozostałych publikacji przed usunięciem subskrypcji, co spowoduje, że menedżer kolejek usunie kolejkę i wszystkie pozostałe publikacje na jej temat.
4. Elastyczne wdrażanie łańcuchów tematów: zarządzanie tematem subskrypcji jest uproszczone przez zdefiniowanie głównej części subskrypcji przy użyciu tematu zdefiniowanego administracyjnie. Część główna drzewa tematów jest następnie ukryta w aplikacji. Ukrywanie części głównej aplikacji można wdrożyć bez potrzeby nieumyślnego utworzenia drzewa tematów, które nakłada się na inne drzewo tematów utworzone przez inną instancję lub inną aplikację.
5. Administrowane tematy: za pomocą łańcucha tematu, w którym pierwsza część jest zgodna z administrowanymi administrowanymi obiektami 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, wówczas subskrypcja może odbierać publikacje od innych elementów klastra.
   - Selektywne dopasowanie zdefiniowanych administracyjnie obiektów tematu i programowo zdefiniowanych subskrypcji umożliwia połączenie korzyści obu z nich. Administrator udostępnia atrybuty dla tematów, a programista dynamicznie definiuje podtematy bez obawy co do zarządzania tematami.
   - Jest to wynikowy łańcuch tematu, który jest używany do dopasowania do obiektu tematu, który udostępnia atrybuty powiązane z tematem, a niekoniecznie obiekt tematu o nazwie w produkcie `sd.Objectname`, chociaż zwykle są one takie same. Patrz “Przykład 2: publikator w temacie o zmiennej” na stronie 905.

Dzięki temu, że subskrypcja jest trwała w tym przykładzie, publikacje będą nadal wysyłane do kolejki subskrypcji po zamknięciu subskrypcji przez subskrybenta za pomocą opcji `MQCO_KEEP_SUB` . Kolejka kontynuuje odbieranie publikacji, gdy subskrybent nie jest aktywny. To zachowanie można przesłonić, tworząc subskrypcję za pomocą opcji `MQSO_PUBLICATIONS_ON_REQUEST` i używając `MQSUBRQ` do żądania zachowanej publikacji.

Subskrypcję można wznowić w późniejszym czasie, otwierając subskrypcję za pomocą opcji `MQCO_RESUME` .

Za pomocą uchwytu kolejki `Hobj`zwróconego przez produkt `MQSUB` na wiele sposobów można użyć uchwytu kolejki. 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 w temacie według właściwości 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 zarządzanej kolejki, otwierając kolejkę zarządzaną po raz drugi przy użyciu zwróconej nazwy kolejki. Kolejka zachowuje się tak, jakby została otwarta do wyłącznego wejścia.

Kolejki niezarządzane są bardziej elastyczne niż kolejki zarządzane. Możliwe jest, na przykład współużytkowanie niezarządzanych kolejek, lub zdefiniowanie wielu subskrypcji w jednej kolejce. W następnym przykładzie przedstawiono sposób łączenia subskrypcji z niezarządzaną kolejką subskrypcji.

Uwaga: Kompaktowy styl kodowania jest przeznaczony do czytelności, a nie do użytku produkcyjnego.

Wyniki są wyświetlane w programie Rysunek 78 na stronie 915.

```
#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 76. Zarządzany subskrybent MQ -część 1: deklaracje i obsługa parametrów.

W tym przykładzie są dostępne dodatkowe komentarze dotyczące deklaracji.

MQHOBJ Hobj = MQHO_NONE;
Nie można jawnie otworzyć nietrwałej zarządzanej kolejki subskrypcji, aby otrzymywać publikacje, ale należy przydzielić pamięć dla uchwytu obiektu, który jest zwracany przez menedżer kolejek po

otwarciu kolejki dla użytkownika. Ważne jest, aby zainicjować uchwyt do produktu MQHO_OBJECT. Wskazuje to na menedżer kolejek, który musi zwrócić uchwyt kolejki do kolejki subskrypcji.

MQSD sd = {MQSD_DEFAULT};
Nowy deskryptor subskrypcji używany w produkcie MQSUB.

MQCHAR48 qName;
Chociaż przykład nie wymaga znajomości kolejki subskrypcji, na przykład zapytanie o nazwę kolejki subskrypcji-powiązanie MQINQ jest nieco niezręczne w języku C, więc może się okazać, że ta część przykładu jest przydatna do badania.

```
    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 77. Zarządzany subskrybent 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 78. Subskrybent produktu MQ

W tym przykładzie są dostępne dodatkowe komentarze dotyczące kodu.

strncpy(sd.ObjectName, topicName, MQ_Q_NAME_LENGTH);
Jeśli parametr topicName ma wartość NULL lub jest pusta *(wartość domyślna)*, nazwa tematu nie jest używana do obliczenia przetłumaczanego łańcucha tematu.

sd.ObjectString.VSPtr = topicString;
Zamiast używać wyłącznie predefiniowanego obiektu tematu, w tym przykładzie programista udostępnia obiekt tematu i łańcuch tematu, które są łączone przez produkt MQSUB. Należy zauważyć, że łańcuch tematu jest strukturą MQCHARV .

sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
Alternatywa dla ustawienia długości pola MQCHARV .

sd.Options = MQSO_CREATE | MQSO_MANAGED | MQSO_NON_DURABLE | MQSO_FAIL_IF_QUIESCING;
Po zdefiniowaniu łańcucha tematu flagi sd.Options wymagają najbardziej uważnej uwagi. Istnieje wiele opcji, na przykład określa się tylko te najczęściej używane. Pozostałe opcje korzystają z wartości domyślnych.

1. Ponieważ subskrypcja jest *nietrwała*, oznacza to, że ma ona czas życia subskrypcji otwartej w aplikacji, należy ustawić flagę MQSO_CREATE . Można również ustawić flagę *(wartość domyślna)* MQSO_NON_DURABLE w celu uzyskania czytelności.
2. Uzupełnieniem produktu MQSO_CREATE jest MQSO_RESUME. Obie opcje mogą być ustawione razem; menedżer kolejek tworzy nową subskrypcję lub wznawia istniejącą subskrypcję, w zależności od tego, która z tych wartości jest odpowiednia. Jeśli jednak zostanie określona opcja MQSO_RESUME , należy również zainicjować strukturę MQCHARV dla produktu sd.SubName, nawet jeśli nie ma subskrypcji, która ma zostać wznowiona. Niepowodzenie inicjowania SubName powoduje zwrócenie kodu powrotu 2440: MQRC_SUB_NAME_ERROR z MQSUB.

   Uwaga: Produkt MQSO_RESUME jest zawsze ignorowany w przypadku nietrwałej subskrypcji zarządzanej: ale określanie jej bez inicjowania struktury MQCHARV dla sd.SubName powoduje błąd.
3. Dodatkowo istnieje trzecia flaga wpływający na sposób otwierania subskrypcji, MQSO_ALTER. Mając na uwadze poprawne uprawnienia, właściwości wznowionej subskrypcji są zmieniane w celu dopasowania do innych atrybutów określonych w MQSUB.

   Uwaga: Należy określić co najmniej jedną z opcji MQSO_CREATE, MQSO_RESUME i MQSO_ALTER . Patrz Opcje (MQLONG). Istnieją przykłady użycia wszystkich trzech flag w produkcie "Przykład 3: Niezarządzany subskrybent MQ" na stronie 916.
4. Ustaw opcję MQSO_MANAGED dla menedżera kolejek, aby automatycznie zarządzać subskrypcją.

sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
Opcjonalnie można pominąć ustawienie długości łańcucha MQCHARV dla łańcuchów zakończonych znakiem NULL i zamiast tego użyć flagi znaku końca wartości NULL.

sd.ResObjectString.VSPtr = resTopicStr;
Wynikowy łańcuch tematu jest umieszczany w programie `printf` w pierwszym miejscu programu. Skonfiguruj program `MQCHARV ResObjectString` for IBM MQ , aby przywrócić rozstrzygnięty łańcuch z powrotem do programu.

Uwaga: Produkt `resTopicStringBuffer` jest inicjowany z wartościami pustymi w produkcie `memset(resTopicStr, 0, sizeof(resTopicStrBuffer))`. Zwrócone łańcuchy tematów nie kończą się znakiem null kończącym.

sd.ResObjectString.VSBufSize = sizeof(resTopicStrBuffer)-1;
Ustaw wielkość buforu `sd.ResObjectString` na jeden mniejszy niż jego wielkość rzeczywista. Zapobiega to nadpisaniu udostępnianego terminatora o wartości NULL, w przypadku gdy rozstrzygnięty łańcuch tematu zapełnia cały bufor.

Uwaga: Jeśli łańcuch tematu jest dłuższy niż `sizeof(resTopicStrBuffer)-1`, nie jest zwracany żaden błąd. Nawet jeśli `VSLength > VSBufSiz` długość zwrócona w polu `sd.ResObjectString.VSLength` jest długością kompletnego łańcucha, a niekoniecznie długością zwróconego łańcucha. Przetestuj `sd.ResObjectString.VSLength < sd.ResObjectString.VSBufSiz` , aby potwierdzić, że łańcuch tematu został zakończony.

MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
Funkcja `MQSUB` tworzy subskrypcję. Jeśli nie jest to trwałe, prawdopodobnie nie interesuje Cię jego nazwa, ale możesz sprawdzić jego status w programie IBM MQ Explorer. Jako dane wejściowe można podać parametr `sd.SubName` , dzięki czemu wiadomo, jaka nazwa będzie wyglądać. Oczywiście należy unikać starć 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. Opcja `MQCLOSE MQCO_REMOVE_SUB` jest domyślna w tym przypadku, ponieważ subskrypcja nie jest trwała. Użycie produktu `MQCO_KEEP_SUB` jest błędem.

Uwaga: Subskrypcja *kolejka* nie jest zamknięta przez produkt `MQSUB`, a jej uchwyt `Hobj`pozostaje poprawny do momentu zamknięcia kolejki przez produkt `MQCLOSE` lub `MQDISC`. Jeśli aplikacja kończy się przedwcześnie, kolejka i subskrypcja są czyszczone przez menedżer kolejek w czasie po zakończeniu aplikacji.

Pojęcia pokrewne

“Przykład 1: konsument publikacji MQ” na stronie 909
Konsument publikowania produktu MQ jest konsumentem komunikatów produktu IBM MQ , który nie subskrybuje się do samych tematów.

“Przykład 3: Niezarządzany subskrybent MQ” na stronie 916
Niezarządzany subskrybent jest ważną klasą aplikacji subskrybenta. Dzięki temu użytkownik łączy korzyści płynące z publikowania/subskrybowania za pomocą *sterowania* kolejkowania i korzystania z publikacji. W przykładzie przedstawiono różne sposoby łączenia subskrypcji i kolejek.

“Zapisywanie aplikacji publikatorów” na stronie 902
Rozpoczęcie pracy z pisaniem aplikacji publikatorów poprzez zapoznanie się z dwoma przykładami. Pierwsza z nich jest możliwie najdokładniej modelowana w punkcie, w którym aplikacja wskazuje na umieszczanie komunikatów w kolejce, a druga demonstruje dynamicznie tworzenie tematów-bardziej powszechny wzorzec dla aplikacji publikatorów.

Przykład 3: Niezarządzany subskrybent MQ
Niezarządzany subskrybent jest ważną klasą aplikacji subskrybenta. Dzięki temu użytkownik łączy korzyści płynące z publikowania/subskrybowania za pomocą *sterowania* kolejkowania i korzystania z publikacji. W przykładzie przedstawiono różne sposoby łączenia subskrypcji i kolejek.

Wzorzec niezarządzany jest zwykle powiązany z *trwałymi* subskrypcjami niż *nietrwały*. Zwykle cykl życia subskrypcji utworzonej przez niezarządzanego subskrybenta jest niezależny od cyklu życia samej aplikacji subskrybującej. Subskrypcja subskrypcji odbiera publikacje nawet wtedy, gdy żadna aplikacja subskrybująca nie jest aktywna.

Można utworzyć trwałe *zarządzane* subskrypcje w celu osiągnięcia tego samego wyniku, 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 stałą kolejkę dla publikacji, które są zgodne z tematem subskrypcji. Powoduje ona usunięcie kolejki i powiązanych z nią publikacji w momencie usunięcia subskrypcji.

Zazwyczaj trwałe *zarządzane* subskrypcje są używane, jeśli cykl życia aplikacji i subskrypcja jest zasadniczo taka sama, ale trudno jest zagwarantować. Dzięki temu, że subskrypcja jest trwała i korzystają z subskrypcji współużytkowanych, każda aplikacja współużytkuje subskrypcję otwierając tę samą kolejkę zarządzaną i pobiera z niej komunikaty.

Subskrypcja *zarządzana* to subskrypcja, w której produkt IBM MQ obsługuje subskrypcję, a następnie rejestruje się i wyrejestrowywać. W subskrypcji *niezarządzanej* aplikacja jest odpowiedzialna za określenie kolejki, w której przechowywane są subskrypcje.

Menedżer kolejek niejawnie otwiera kolejkę subskrypcji trwałej dla subskrybenta w taki sposób, że współużytkowane przetwarzanie kolejki nie jest możliwe. Ponadto nie można utworzyć więcej niż jednej subskrypcji dla każdej zarządzanej kolejki, a kolejki są trudniejsze do zarządzania, ponieważ użytkownik ma mniejszą kontrolę nad nazwami kolejek. Z tych powodów należy rozważyć, czy *niezarządzany* subskrybent produktu MQ jest lepiej dopasowany do aplikacji wymagających trwałych subskrypcji niż subskrybent *zarządzany* MQ .

Kod w produkcie Rysunek 81 na stronie 923 demonstruje niezarządzany wzorzec trwałej subskrypcji. W przypadku ilustracji kod tworzy również subskrypcje niezarządzane, nietrwałe. W tym przykładzie przedstawiono następujące aspekty wzorca:

- Subskrypcje na żądanie: łańcuchy tematów subskrypcji są dynamiczne. Są one udostępniane przez aplikację po jej uruchomieniu.
- Uproszczone zarządzanie tematem subskrypcji: zarządzanie tematem subskrypcji jest uproszczone przez zdefiniowanie głównej części łańcucha tematu subskrypcji przy użyciu tematu zdefiniowanego administracyjnie. Powoduje to ukrycie głównej części drzewa tematów z aplikacji. Ukrywanie części głównej subskrybenta może być wdrażane w różnych drzewach tematów.
- Elastyczne zarządzanie subskrypcją: można zdefiniować subskrypcję administracyjnie lub utworzyć ją na żądanie w ramach programu subskrybenta. Nie ma różnicy między administracyjnym i programowo utworzonym subskrypcją, z wyjątkiem atrybutu, który pokazuje, w jaki sposób subskrypcja została utworzona. Istnieje trzeci typ subskrypcji, który jest tworzony automatycznie przez menedżer kolejek w celu dystrybucji subskrypcji. Wszystkie subskrypcje są wyświetlane w Eksploratorze IBM MQ .
- Elastyczne powiązanie subskrypcji z kolejkami: predefiniowana kolejka lokalna jest powiązana z subskrypcją za pomocą funkcji MQSUB . Istnieją różne sposoby korzystania z programu MQSUB w celu powiązania subskrypcji z kolejkami:
  - Powiązanie subskrypcji z kolejką o *nie* istniejących subskrypcjach `MQSO_CREATE + (Hobj from MQOPEN)`.
  - Powiąż *nową* subskrypcję z kolejką, która ma istniejące subskrypcje, `MQSO_CREATE + (Hobj from MQOPEN)`.
  - Przenieś istniejącą subskrypcję do innej kolejki, `MQSO_ALTER + (Hobj from MQOPEN)`.
  - Wznów istniejącą subskrypcję powiązaną z istniejącą kolejką, `MQSO_RESUME + (Hobj = MQHO_NONE)` lub `MQSO_RESUME + (Hobj = from MQOPEN of queue with existing subscription)`.
  - Łącząc produkt `MQSO_CREATE | MQSO_RESUME | MQSO_ALTER` w różnych kombinacjach, można używać różnych stanów wejściowych subskrypcji i kolejki bez konieczności tworzenia kodu wielu wersji produktu MQSUB z różnymi wartościami `sd.Options` .
  - Alternatywnie, poprzez kodowanie konkretnego wyboru `MQSO_CREATE | MQSO_RESUME | MQSO_ALTER` , menedżer kolejek zwraca błąd (Tabela 127 na stronie 919) Jeśli stany subskrypcji i kolejki wprowadzone jako dane wejściowe dla MQSUB są niespójne z wartością `sd.Options`. W programie Rysunek 87 na stronie 926 wyświetlane są wyniki wydania MQSUB dla subskrypcji X z różnymi ustawieniami opcji `sd.Options` i przekazywanie jej trzech różnych uchwytów obiektów.

Zapoznaj się z różnymi danymi wejściowymi do przykładowego programu w programie Rysunek 80 na stronie 921 , aby zapoznać się z różnymi rodzajami błędów. Jeden wspólny błąd, RC = 2440, który nie jest zawarty w przypadkach wymienionych w tabeli, jest błędem nazwy subskrypcji. Jest to zwykle spowodowane przez przekazanie nazwy subskrypcji o wartości NULL lub niepoprawnej za pomocą produktu MQSO_RESUME lub MQSO_ALTER.

- Multiprocessing: Możesz podzielić się wśród wielu konsumentów pracą lektur publikacji. Wszystkie publikacje znajdują się w pojedynczej kolejce powiązanej z tematem subskrypcji. Konsumenci mają możliwość bezpośredniego otwierania kolejki przy użyciu produktu MQOPEN lub wznawiania subskrypcji przy użyciu produktu MQSUB.
- Koncentracja subskrypcji: w tej samej kolejce może zostać utworzonych wiele subskrypcji. Należy zachować ostrożność przy użyciu tej możliwości, ponieważ może ona prowadzić do nakładania się subskrypcji, a także wielokrotnie odbierać tę samą publikację. Opcja MQSO_GROUP_SUB eliminuje zduplikowane publikacje spowodowane nakładającymi się subskrypcjami.
- Abonent i separacja konsumentów: Tak jak w przypadku trzech modeli konsumenckich przedstawionych na przykładach, inny model polega na oddzieleniu konsumenta od abonenta. Jest to odmiana niezarządzanego subskrybenta MQ , ale zamiast wydawania produktów MQOPEN i MQSUB w tym samym programie, jeden program subskrybuje publikacje, a inny program zużywa je. Subskrybent może na przykład należeć do klastra publikowania/subskrybowania, a konsument przyłączony do menedżera kolejek poza klastrem menedżera kolejek. Konsument otrzymuje publikacje za pośrednictwem standardowego rozproszonego kolejkowania, definiując kolejkę subskrypcji jako definicję kolejki zdalnej.

Zrozumienie zachowania produktu MQSO_CREATE | MQSO_RESUME | MQSO_ALTER jest ważne, zwłaszcza jeśli planowane jest uproszczenie kodu za pomocą kombinacji tych opcji. Należy zbadać tabelę Tabela 127 na stronie 919 , która przedstawia wyniki przekazywania różnych uchwytów kolejek do tabeli MQSUB, a także wyniki działania przykładowego programu przedstawionego w programie Rysunek 82 na stronie 924 na wartość Rysunek 87 na stronie 926.

Scenariusz użyty do skonstruowania tabeli ma jedną subskrypcję X i dwie kolejki, A i B. Parametr nazwy subskrypcji sd.SubName jest ustawiony na wartość X(nazwa subskrypcji przyłączonej do kolejki A). W kolejce B nie jest przyłączona żadna subskrypcja.

W produkcie Tabela 127 na stronie 919 MQSUB jest przekazywana subskrypcja X i uchwyt kolejki do kolejki A. Wyniki z opcji subskrypcji są następujące:

- MQSO_CREATE nie powiodło się, ponieważ uchwyt kolejki odpowiada kolejce A , która ma już subskrypcję X. To zachowanie jest przeciwstawne do pomyślnego wywołania. Wywołanie to powiodło się, ponieważ kolejka B nie ma subskrypcji do X dołączonej do niej.
- Pomyślnie MQSO_RESUME , ponieważ uchwyt kolejki odpowiada kolejce A , która ma już subskrypcję X. W przeciwieństwie do tego wywołanie nie powiedzie się, jeśli subskrypcja X nie istnieje w kolejce A.
- Produkt MQSO_ALTER działa w podobny sposób, jak produkt MQSO_RESUME w celu otwarcia subskrypcji i kolejki. Jeśli jednak atrybuty zawarte w deskryptorze subskrypcji przekazanej do produktu MQSUB różnią się od atrybutów subskrypcji, działanie produktu MQSO_RESUME nie powiedzie się, natomiast operacja MQSO_ALTER zakończy się tak długo, jak długo instancja programu ma uprawnienia do zmiany atrybutów. Należy pamiętać, że nigdy nie można zmienić łańcucha tematu w subskrypcji. Zamiast zwrócić błąd, program MQSUB ignoruje nazwę tematu i wartości łańcucha tematu w deskryptorze subskrypcji i używa wartości w istniejącej subskrypcji.

Następnie należy sprawdzić Tabela 127 na stronie 919 , gdzie zmaterializowana tabela zapytania (MQSUB) jest przekazywana do subskrypcji X, a uchwyt kolejki do kolejki B. Wyniki z opcji subskrypcji są następujące:

- MQSO_CREATE udaje się i tworzy subskrypcję X w kolejce B , ponieważ jest to nowa subskrypcja w kolejce B.
- MQSO_RESUME nie powiodło się. Produkt MQSUB wyszukuje subskrypcję X w kolejce B i nie znajduje jej, ale nie zwraca wartości *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. Produkt MQSUB oczekuje, że uchwyt kolejki będzie wskazywał kolejkę z subskrypcją. Sprawdza to najpierw przed sprawdzeniem, czy subskrypcja o nazwie sd.SubName istnieje.

- MQSO_ALTER zakończy się powodzeniem, a następnie przenosi subskrypcję z kolejki A do kolejki B.

Przypadek, który nie jest wyświetlany w tabeli, jest taki, że jeśli nazwa subskrypcji subskrypcji w kolejce A nie jest zgodna z nazwą subskrypcji w produkcie sd.SubName. Wywołanie to kończy się niepowodzeniem z kodem powrotu *RC = 2428-subskrypcja X nie istnieje w kolejce A*.

Tabela 127. Błędy z tabeli MQSUB z różnymi uchwytami kolejek i kombinacjami subskrypcji

Uchwyty kolejek	**Kolejka A** **Subskrypcja X** **Kolejka B** **Brak subskrypcji**	**Kolejka A** **Brak subskrypcji** **Kolejka B** **Brak subskrypcji**
Hobj dla kolejki *Kolejka A* przekazanej do tabeli MQSUB	**MQSO_CREATE** RC = 2432-Subskrypcja X już istnieje w kolejce A **MQSO_RESUME** Wznawia subskrypcję X w kolejce A **MQSO_ALTER** Wznawia subskrypcję X w kolejce A i wprowadza dozwolone zmiany	**MQSO_CREATE** Tworzy subskrypcję X w kolejce A **MQSO_RESUME** RC = 2428-Subskrypcja X nie istnieje w kolejce A **MQSO_ALTER** RC = 2428-Subskrypcja X nie istnieje w kolejce A
Hobj dla kolejki *Kolejka B* przekazany do tabeli MQSUB	**MQSO_CREATE** Tworzy nową subskrypcję X w kolejce B **MQSO_RESUME** RC = 2019-Kolejka subskrypcji nie jest zgodna z uchwytem obiektu kolejki **MQSO_ALTER** Przenieś subskrypcję X z kolejki A do kolejki B	**MQSO_CREATE** Tworzy nową subskrypcję X w kolejce B **MQSO_RESUME** RC = 2428-subskrypcja X nie istnieje w kolejce B **MQSO_ALTER** RC = 2428-subskrypcja X nie istnieje w kolejce B
MQHO_NONE przekazano do tabeli MQSUB	**MQSO_CREATE** RC = 2019-Bad object handle: set MQSO_MANAGED flag to create a managed subscription and create a managed queue **MQSO_RESUME** Wznawia subskrypcję X w kolejce A i zwraca Hobj do kolejki A **MQSO_ALTER** Wznawia subskrypcję X w kolejce A, zwraca Hobj do kolejki A i wprowadza dozwolone zmiany	**MQSO_CREATE** RC = 2019-Bad object handle: set MQSO_MANAGED flag to create a managed subscription and create a managed queue **MQSO_RESUME** RC = 2428-Brak subskrypcji X **MQSO_ALTER** RC = 2019-Błędny uchwyt obiektu: brak kolejki A lub B

Uwaga: Kompaktowy 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 79. Niezarządzany subskrybent 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 80. Niezarządzany subskrybent MQ -część 2: obsługa parametrów.

Dodatkowe komentarze dotyczące obsługi parametrów w tym przykładzie są następujące:

switch((argv[5][0]))
Istnieje możliwość wprowadzenia parametru A lter | C reate | R esume w parametrze 5, aby przetestować efekt przesłonięcie części ustawienia opcji MQSUB , która jest używana domyślnie w tym przykładzie. Domyślnym ustawieniem używanym w tym przykładzie jest MQSO_CREATE | MQSO_RESUME | MQSO_DURABLE.

Uwaga: Ustawienie MQSO_ALTER lub MQSO_RESUME bez ustawienia MQSO_DURABLE jest błędem, a sd.SubName musi być ustawione i odwoływać się do subskrypcji, która może zostać wznowiona lub zmieniona.

***subscriptionQueue = '\0';**
sdOptions = sdOptions + MQSO_MANAGED;
Jeśli domyślna kolejka subskrypcji STOCKTICKER jest zastępowana łańcuchem pustym, o ile ustawiona jest wartość MQSO_CREATE , w przykładzie ustawiana jest flaga MQSO_MANAGED i tworzona

jest kolejka subskrypcji dynamicznej. Jeśli parametr `Alter or Resume` jest ustawiony w piątym parametrze, zachowanie tego przykładu będzie zależeć od wartości `subscriptionName`.

```
*subscriptionName = '\0';
sdOptions = sdOptions - MQSO_DURABLE;
```

Jeśli domyślna subskrypcja, `IBMSTOCKPRICESUB`, zostanie zastąpiona łańcuchem pustym, na przykład zostanie usunięta flaga `MQSO_DURABLE` . Jeśli zostanie uruchomiony przykład udostępniający wartości domyślne dla innych parametrów, zostanie utworzona dodatkowa subskrypcja tymczasowa przeznaczona dla produktu `STOCKTICKER` , a następnie zostaną odebrane zduplikowane publikacje. Następnym razem, gdy uruchomisz przykład, bez żadnych parametrów, otrzymasz po prostu jedną publikację.

```
    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 81. Niezarządzany subskrybent MQ -część 3: treść kodu.

Dodatkowe komentarze na temat kodu w tym przykładzie są następujące:

if (strlen(subscriptionQueue))
Jeśli nie istnieje nazwa kolejki subskrypcji, wówczas w przykładzie użyto wartości MQHO_NONE jako wartości Hobj.

MQOPEN(...);
Kolejka subskrypcji zostanie otwarta, a uchwyt kolejki został zapisany w produkcie Hobj.

MQSUB(Hconn, &sd, &Hobj, &Hsub, &CompCode, &Reason);
Subskrypcja jest otwierana za pomocą Hobj przekazanego z MQOPEN (lub MQHO_NONE , jeśli nie ma nazwy kolejki subskrypcji). Kolejka niezarządzana może zostać wznowiona bez jawnego otwierania go za pomocą MQOPEN.

MQCLOSE(Hconn, &Hsub, MQCO_NONE, &CompCode, &Reason);
Subskrypcja jest zamknięta za pomocą uchwytu subskrypcji. W zależności od tego, czy subskrypcja jest trwała, czy nie, subskrypcja jest zamykana z niejawnym MQCO_KEEP_SUB lub MQCO_REMOVE_SUB. Istnieje możliwość zamknięcia trwałej subskrypcji za pomocą produktu MQCO_REMOVE_SUB, ale nie można zamknąć nietrwałej subskrypcji za pomocą produktu MQCO_KEEP_SUB. Działanie produktu MQCO_REMOVE_SUB polega na usunięciu subskrypcji, w której zatrzymywane są dalsze publikacje wysyłane do kolejki subskrypcji.

MQCLOSE(Hconn, &Hobj, MQCO_NONE, &CompCode, &Reason);
Nie jest podejmowane żadne specjalne działanie, jeśli subskrypcja jest niezarządzana. Jeśli kolejka jest zarządzana, a subskrypcja została zamknięta z jawnym lub niejawnym MQCO_REMOVE_SUB, to wszystkie publikacje są usuwane z kolejki i kolejki usunięte w tym momencie.

gmo.MatchOptions = MQMO_MATCH_CORREL_ID;
memcpy(md.CorrelId, sd.SubCorrelId, MQ_CORREL_ID_LENGTH);
Upewnij się, że otrzymane komunikaty są tymi, które są przeznaczone dla subskrypcji.

Wyniki przykładu ilustrują aspekty publikowania/subskrypcji:

W produkcie Rysunek 82 na stronie 924 przykład rozpoczyna się publikowaniem 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 82. Opublikuj od 130 do NYSE/IBM/PRICE

W programie Rysunek 83 na stronie 924 wykonanie przykładu z użyciem parametrów domyślnych odbiera zachowaną publikację 130. Podany obiekt tematu i łańcuch tematu są ignorowane, jak to pokazano w sekcji Rysunek 87 na stronie 926. Obiekt tematu i łańcuch tematu są zawsze pobierane z obiektu subskrypcji, gdy jest on udostępniany, a łańcuch tematu jest niezmienny. Rzeczywiste zachowanie przykładu zależy od wyboru lub kombinacji produktów MQSO_CREATE, MQSO_RESUMEi MQSO_ALTER. W tym przykładzie MQSO_RESUME jest wybraną opcją.

```
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 83. Odbierz zachowaną publikację

W (Rysunek 84 na stronie 925) nie otrzymano żadnych publikacji, ponieważ trwała subskrypcja otrzymała już zachowaną publikację. W tym przykładzie subskrypcja jest wznawiana przez podanie tylko nazwy subskrypcji bez nazwy kolejki. Jeśli podana nazwa kolejki została podana, kolejka zostanie otwarta jako pierwsza, a uchwyt przekazany do programu MQSUB.

Uwaga: The 2038 error from MQINQ is due to the implicit MQOPEN of STOCKTICKER by MQSUB not including the MQOO_INQUIRE option. Należy unikać kodu powrotu produktu 2038 z programu MQINQ , otwierając jawnie 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 84. Wznów subskrypcję

W produkcie Rysunek 85 na stronie 925przykład tworzy nietrwałą subskrypcję niezarządzaną za pomocą programu 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 85. Otrzymuj zachowaną publikację z nową, niezarządzaną subskrypcją nietrwałą

W programie Rysunek 86 na stronie 925, aby zademonstrować nakładające się subskrypcje, wysyłana jest inna publikacja, która zmienia zachowaną publikację. Następnie tworzona jest nowa, nietrwała, niezarządzana subskrypcja, która nie udostępnia nazwy subskrypcji. Zachowana publikacja jest otrzymywana dwa razy, raz dla nowej subskrypcji, a raz dla trwałej subskrypcji produktu IBMSTOCKPRICESUB , która jest nadal aktywna w kolejce produktu STOCKTICKER . Przykładem jest ilustracja, że jest to kolejka, która ma subskrypcje, a nie aplikacja. Mimo że nie nawiązano do subskrypcji produktu IBMSTOCKPRICESUB w tym wywołaniu aplikacji, aplikacja otrzymuje publikację dwa razy: jeden raz od trwałej subskrypcji, która została utworzona administracyjnie, a raz z nietrwałej subskrypcji utworzonej 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 86. Nakładające się subskrypcje

W produkcie Rysunek 87 na stronie 926 przykład demonstruje, że udostępnienie nowego łańcucha tematu i istniejącej subskrypcji nie powoduje zmiany subskrypcji.

1. W pierwszym przypadku program `Resume` wznawia istniejącą subskrypcję, czego można się spodziewać, a także ignoruje zmieniony łańcuch tematu.
2. W drugim przypadku produkt `Alter` powoduje wystąpienie błędu `RC = 2510, Topic not alterable`.
3. W trzecim przykładzie program `Create` powoduje wystąpienie błędu `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 87. Tematy subskrypcji nie mogą być zmieniane

Pojęcia pokrewne

“Przykład 1: konsument publikacji MQ” na stronie 909
Konsument publikowania produktu MQ jest konsumentem komunikatów produktu IBM MQ , który nie subskrybuje się do samych tematów.

“Przykład 2: zarządzany subskrybent produktu MQ” na stronie 911
Zarządzany subskrybent MQ jest preferowanym wzorcem dla większości aplikacji subskrybentów. W tym przykładzie wymagana jest *nie* administracyjna definicja kolejek, tematów lub subskrypcji.

“Zapisywanie aplikacji publikatorów” na stronie 902
Rozpoczęcie pracy z pisaniem aplikacji publikatorów poprzez zapoznanie się z dwoma przykładami. Pierwsza z nich jest możliwie najdokładniej modelowana w punkcie, w którym aplikacja wskazuje na umieszczanie komunikatów w kolejce, a druga demonstruje dynamicznie tworzenie tematów-bardziej powszechny wzorzec dla aplikacji publikatorów.

Cykle życia publikowania/subskrybowania

W projektowaniu aplikacji publikowania/subskrypcji należy wziąć pod uwagę cykle życia tematów, subskrypcji, subskrybentów, publikacji, publikatorów i kolejek.

Cykl życia obiektu, taki jak subskrypcja, rozpoczyna się od jego utworzenia, a kończy wraz z jego usunięciem. Może również zawierać inne stany i zmiany, które przechodzi, takie jak tymczasowe zawieszenie, które mają tematy nadrzędne i podrzędne, są przedawane i usuwane.

Tradycyjnie IBM MQ obiekty, takie jak kolejki, są tworzone administracyjnie lub przez programy administracyjne przy użyciu formatu komend programowalnych (PCF). Publikowanie/subskrypcja jest inne w przypadku udostępniania komend API MQSUB i MQCLOSE w celu tworzenia i usuwania subskrypcji, mając pojęcie zarządzanych subskrypcji, które nie tylko tworzą i usuwają kolejki, ale także usuwają niewykorzystane komunikaty i mają powiązania między obiektami tematu utworzonymi administracyjnie i programowo lub administracyjnie utworzonymi łańcuchami tematów.

To funkcjonalne gąsienice bogactwa funkcjonalnego dla szerokiego zakresu wymagań publikowania/subskrypcji, a także upraszcza projektowanie niektórych wspólnych wzorców aplikacji publikowania/subskrypcji. Zarządzane subskrypcje, na przykład, upraszczają programowanie i administrowanie subskrypcją, która ma trwać tylko tak długo, jak program, który go utworzył. Subskrypcje niezarządzane upraszczają programowanie w przypadku, gdy istnieje luźniejsze połączenie między publikacjami zasubskrybowanych i konsumowanych. Centralnie utworzone subskrypcje są przydatne w przypadku, gdy wzorzec jest jednym z ruchu publikacji routingu skierowanym do konsumentów na podstawie scentralizowanego modelu sterowania, na przykład wysyłania informacji o locie do zautomatyzowanych bram, podczas gdy programowo tworzone subskrypcje mogą być używane, jeśli personel bramy jest odpowiedzialny za subskrybowanie rekordów pasażerów dla tego lotu, poprzez wprowadzenie numeru lotu w bramce.

W tym ostatnim przykładzie może być odpowiednia zarządzana subskrypcja trwała: zarządzane, ponieważ subskrypcje są tworzone bardzo często i mają czytelny punkt końcowy, gdy brama jest zamykana, a subskrypcja może być programowo usunięta; trwała, aby uniknąć utraty rekordu pasażera ze względu na to, że subskrybent bramy jest wyłączony z jednego lub innego powodu.[8]. W celu zainicjowania publikacji zapisów pasażerów do bramy, ewentualny projekt będzie dla aplikacji bramowej obu subskrybować

[8] Wydawca musi wysłać rekordy pasażerów jako komunikaty trwałe, aby uniknąć innych możliwych awarii, oczywiście.

rekordy pasażerów przy użyciu numeru bramy, i publikuje zdarzenie otwarcia bramy z wykorzystaniem numeru bramy. Wydawca odpowiada na zdarzenie otwarcia bram, publikując zapisy pasażerów-które mogą następnie trafić również do innych zainteresowanych, takich jak fakturowanie, do rejestrowania lotu, a także do obsługi klienta, do powiadomień tekstowych na telefony komórkowe pasażerskie o numerze bram.

Centralnie zarządzana subskrypcja może używać trwałego modelu niezarządzanego, kierujących listy pasażerów do bramy przy użyciu predefiniowanej kolejki dla każdej bramy.

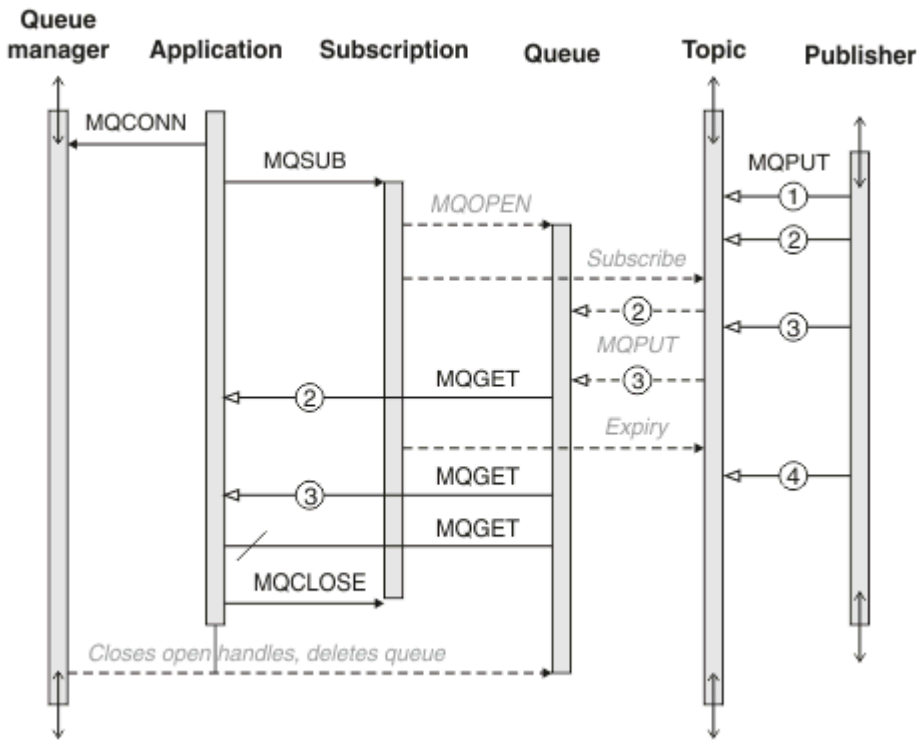
Następujące trzy przykłady cykli życia publikowania/subskrypcji ilustrują sposób, w jaki zarządzane są nietrwałe, zarządzane trwałe i niezarządzane, trwałe subskrybenci współdziałają z subskrypcjami, tematami, kolejkami, publikatorami i menedżerem kolejek oraz w jaki sposób odpowiedzialność może być podzielona między administrację i programy subskrybentów.

Zarządzany nietrwały subskrybent

Program Rysunek 88 na stronie 928 wyświetla aplikację tworzący zarządzane nietrwałe subskrypcje, pobierając dwa komunikaty publikowane w temacie określonym w subskrypcji i kończące się. Interakcje oznaczone kursywą szarą czcionką z strzałkami w kropce są niejawne.

Są pewne punkty, które warto zwrócić uwagę.

1. Aplikacja tworzy subskrypcję w temacie, który został już opublikowany dwukrotnie. Po otrzymaniu pierwszej publikacji subskrybent otrzymuje publikację *sekunda* , która jest aktualnie zachowaną publikacją.
2. Menedżer kolejek tworzy tymczasową kolejkę subskrypcji, a także tworzy subskrypcję dla tego tematu.
3. Subskrypcja ma limit czasu utraty ważności. Jeśli subskrypcja utraci ważność, nie będą wysyłane do tej subskrypcji żadne publikacje dotyczące tego tematu, ale subskrybent będzie nadal otrzymując komunikaty publikowane przed utratą ważności subskrypcji. Utrata ważności subskrypcji nie ma wpływu na wygaśnięcie subskrypcji.
4. Czwarta publikacja nie jest umieszczana w kolejce subskrypcji i w związku z tym ostatnia publikacja `MQGET` nie zwraca publikacji.
5. Mimo że subskrybent zamknie subskrypcję, nie zamyka połączenia z kolejką lub menedżerem kolejek.
6. Menedżer kolejek czyści wkrótce po zakończeniu działania aplikacji. Ponieważ subskrypcja jest zarządzana i nie jest trwała, kolejka subskrypcji jest usuwana.



Rysunek 88. Zarządzane nietrwałe wąwozy subskrybentów

Zarządzany trwały subskrybent

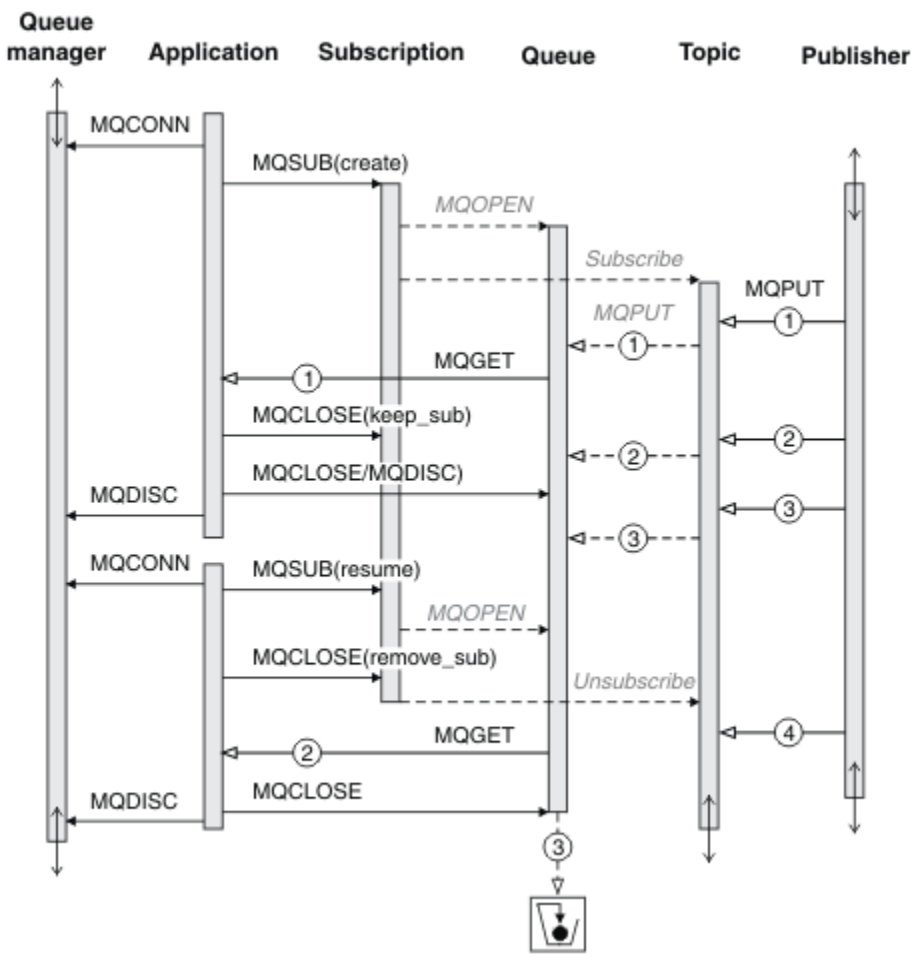
Zarządzany trwały subskrybent ma następny przykład krok dalej, a subskrypcja zarządzana zachowa zakończenie i restartowanie aplikacji subskrybującej.

Istnieją nowe punkty do odnoenia.

1. W tym przykładzie, w przeciwieństwie do ostatniego, temat publikowania nie istniał przed zdefiniowaniem go w subskrypcji.
2. Gdy subskrybent zostanie zamknięty po raz pierwszy, subskrypcja zostanie zamknięta z opcją MQCO_KEEP_SUB. To jest zachowanie domyślne w przypadku niejawnego zamykania zarządzanej subskrypcji trwałej.
3. Gdy subskrybent wznawia subskrypcję, kolejka subskrypcji jest ponownie otwierana.
4. Nowa publikacja 2, umieszczona w kolejce przed jej ponownym otwarciom, jest dostępna dla produktu MQGET, nawet po usunięciu subskrypcji.

   Mimo że subskrypcja jest trwała, subskrybent niezawodnie odbiera wszystkie wiadomości wysłane przez publikatora tylko wtedy, gdy *obie* subskrypcje są trwałe, a komunikaty trwałe. Trwałość komunikatu zależy od ustawienia pola Persistent w MQMD komunikatu wysłanego przez publikatora. Abonent nie ma nad tym żadnej kontroli.
5. Zamknięcie subskrypcji z flagą MQCO_REMOVE_SUB powoduje usunięcie subskrypcji, zatrzymanie kolejnych publikacji umieszczonych w kolejce subskrypcji. Gdy kolejka subskrypcji jest zamknięta, menedżer kolejek usuwa nieprzeczytaną publikację 3, a następnie usuwa kolejkę. Czynność jest równoważna administracyjnie usuwaniu subskrypcji.

   Uwaga: Nie należy usuwać kolejki ręcznie lub MQCLOSE z opcją MQCO_DELETElub MQCO_PURGE_DELETE. Szczegółowe informacje o implementacji subskrypcji zarządzanej nie są częścią obsługiwanego interfejsu IBM MQ . Menedżer kolejek nie może zarządzać subskrypcją niezawodnie, chyba że ma pełną kontrolę.



Rysunek 89. Zarządzane trwałe elementy życia subskrybentów

Niezarządzany trwały subskrybent

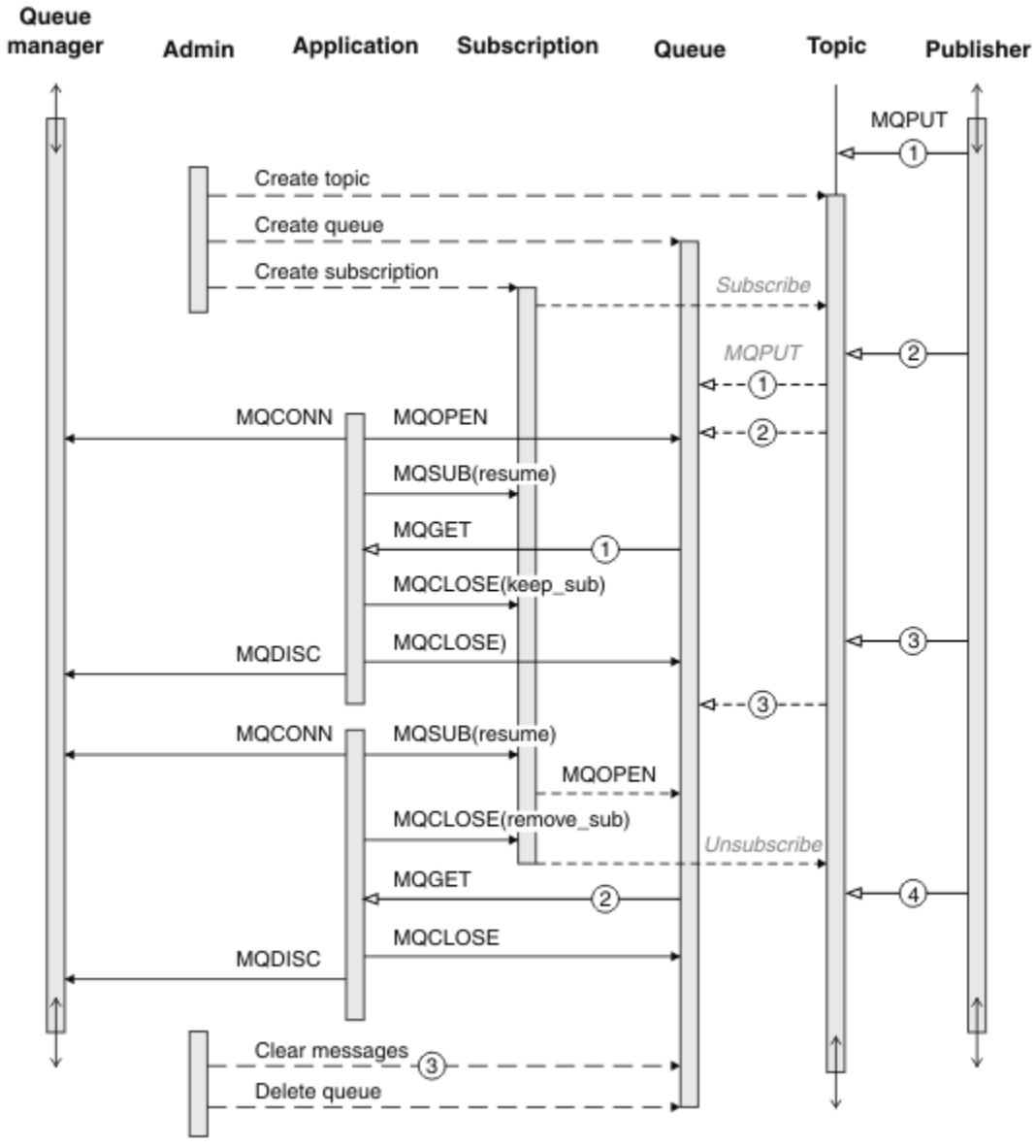
W trzecim przykładzie zostanie dodany administrator: niezarządzany trwały subskrybent. Dobrym przykładem jest pokazanie, w jaki sposób administrator może wchodzić w interakcje z aplikacją publikowania/subskrypcji.

Punkty do nota są wymienione.

1. Publikator umieszcza komunikat 1w temacie, który później staje się powiązany z obiektem tematu, który jest używany do subskrypcji. Obiekt tematu definiuje łańcuch tematu, który jest zgodny z tematem, który został opublikowany przy użyciu znaków wieloznacznych.
2. W temacie znajduje się zachowana publikacja.
3. Administrator tworzy obiekt tematu, kolejkę i subskrypcję. Obiekt tematu i kolejka muszą zostać zdefiniowane przed subskrypcją.
4. Aplikacja otwiera kolejkę powiązaną z subskrypcją i przekazuje `MQSUB` uchwyt kolejki. Może ona, alternatywnie, po prostu otworzyć subskrypcję, przekazując jej uchwyt kolejki `MQHO_NONE`. Konwersacja nie jest prawdziwa, nie można wznowić subskrypcji, przekazując jej tylko uchwyt kolejki bez nazwy subskrypcji-kolejka może mieć wiele subskrypcji.
5. Aplikacja otwiera subskrypcję przy użyciu opcji `MQSO_RESUME` , mimo że po raz pierwszy otwarto subskrypcję. Wznawiana jest administracyjna subskrypcja utworzona przez administratora.

6. Subskrybent otrzymuje zachowaną publikację, 1. Publikacja 2, mimo że została opublikowana przed otrzymaniem publikacji przez subskrybenta, została opublikowana po rozpoczęciu subskrypcji i jest drugą publikacją w kolejce subskrypcji.

   Uwaga: Jeśli zachowana publikacja nie zostanie opublikowana jako komunikat trwały, zostanie ona utracona po zrestartowaniu menedżera kolejek.
7. W tym przykładzie subskrypcja jest trwała. Program może utworzyć niezarządzaną, nietrwałą subskrypcję. To oczywiste, że nie jest to coś, co może zrobić administrator.
8. Efektem działania opcji MQCO_REMOVE_SUB przy zamykaniu subskrypcji jest usunięcie subskrypcji tak, jakby jej administrator usunował tę subskrypcję. Spowoduje to zatrzymanie kolejnych publikacji wysyłanych do kolejki, ale nie ma wpływu na publikacje, które są już w kolejce, nawet jeśli kolejka jest zamknięta, w przeciwieństwie do trwałej subskrypcji *zarządzanej* .
9. Później administrator usunie pozostały komunikat 3i usunie kolejkę.



Rysunek 90. Niezarządzane trwałe elementy życia subskrybentów

Normalny wzorzec dla subskrypcji niezarządzanej jest przeznaczony do przechowywania w kolejce i subskrypcji, który ma być wykonywany przez administratora. Zwykle nie jest podejmowana próba

emulowania zachowania zarządzanego subskrybenta oraz programowego programowania kolejek i subskrypcji w kodzie aplikacji. Jeśli użytkownik musi napisać logikę zarządzania, należy sprawdzić, czy możliwe jest osiągnięcie tych samych wyników przy użyciu wzorca zarządzanego. Nie jest łatwo pisać ściśle zsynchronizowany, całkowicie niezawodny kod zarządzania. Łatwiejsze jest późniejsze przechyłowanie, ręcznie lub za pomocą zautomatyzowanego programu do zarządzania, kiedy można mieć pewność, że wiadomości, subskrypcje i kolejki mogą być po prostu usunięte, niezależnie od ich stanu.

Właściwości komunikatu publikowania/subskrypcji

Wiele właściwości komunikatu jest powiązanych z przesyłaniem komunikatów publikowania/subskrypcji produktu IBM MQ .

Znacznik PubAccounting

Jest to wartość, która będzie znajdować się w polu AccountingToken deskryptora komunikatu (MQMD) wszystkich komunikatów publikacji zgodnych z tą subskrypcją. Element AccountingToken jest częścią kontekstu tożsamości komunikatu. Więcej informacji na temat kontekstu komunikatów zawiera sekcja “Kontekst komunikatu” na stronie 47. Więcej informacji na temat pola AccountingToken w strukturze MQMD można znaleźć w sekcji AccountingToken.

PubApplIdentityData

Jest to wartość, która będzie w polu ApplIdentityData deskryptora komunikatu (MQMD) wszystkich komunikatów publikacji zgodnych z tą subskrypcją. ApplIdentityDane są częścią kontekstu tożsamości komunikatu. Więcej informacji na temat kontekstu komunikatów zawiera sekcja “Kontekst komunikatu” na stronie 47. Więcej informacji na temat pola ApplIdentityData w strukturze MQMD zawiera sekcja ApplIdentityDane.

Jeśli opcja MQSO_SET_IDENTITY_CONTEXT nie zostanie podana, dane ApplIdentity, które zostaną ustawione w każdym komunikacie opublikowanym dla tej subskrypcji, są puste, jako domyślne informacje o kontekście.

Jeśli zostanie podana opcja MQSO_SET_IDENTITY_CONTEXT, użytkownik PubApplIdentityData jest generowany przez użytkownika, a pole to zawiera pole wejściowe zawierające dane ApplIdentity, które mają być ustawione w każdej publikacji dla tej subskrypcji.

PubPriority

Jest to wartość, która będzie należeć do pola 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-W przypadku, gdy kolejka subskrypcji jest udostępniana w polu Hobj w wywołaniu MQSUB, a nie jest zarządzana, priorytet dla komunikatu jest przyjmowany z atrybutu DefProirity tej kolejki. Jeśli identyfikowana 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 polu Priorytet w strukturze MQMD. Jeśli wywołanie MQSUB korzysta z uchwytu zarządzanego, priorytet komunikatu jest przyjmowany z atrybutu DefPriority w kolejce modelowej powiązanej z subskrybowanym tematem.
- MQPRI_PRIORITY_AS_PUBLISHED-priorytet dla komunikatu jest priorytetem pierwotnej publikacji. Jest to początkowa wartość tego pola.

Identyfikator SubCorrel

Ostrzeżenie: Identyfikator korelacji może być przekazywany tylko między menedżerami kolejek w klastrze publikowania/subskrypcji, a nie w hierarchii.

Wszystkie publikacje wysłane w celu dopasowania do tej subskrypcji będą zawierać ten identyfikator korelacji w deskryptorze komunikatu. Jeśli wiele subskrypcji korzysta z tej samej kolejki w celu pobrania ich publikacji, użycie identyfikatora MQGET według identyfikatora korelacji umożliwia uzyskanie tylko tych publikacji, które mają zostać uzyskane. Ten identyfikator korelacji może być wygenerowany przez menedżera kolejek lub przez użytkownika.

Jeśli opcja MQSO_SET_CORREL_ID nie jest określona, identyfikator korelacji jest generowany przez menedżer kolejek, a pole to jest polem wyjściowym zawierającym identyfikator korelacji, który zostanie ustawiony w każdym komunikacie opublikowanym dla tej subskrypcji.

Jeśli zostanie podana opcja MQSO_SET_CORREL_ID, identyfikator korelacji jest generowany przez użytkownika, a pole to jest polem wejściowym zawierającym identyfikator korelacji, który ma być ustawiony w każdej publikacji dla tej subskrypcji. W takim przypadku, jeśli pole zawiera wartość MQCI_NONE, to identyfikator korelacji, który zostanie ustawiony w każdym komunikacie opublikowanym dla tej subskrypcji, będzie identyfikatorem korelacji utworzonym przez oryginalną nazwę komunikatu.

Jeśli określona jest opcja MQSO_GROUP_SUB, a określony identyfikator korelacji jest taki sam, jak istniejąca grupowa subskrypcja za pomocą tej samej kolejki i nakładającego się łańcucha tematu, tylko najbardziej znacząca subskrypcja w grupie jest udostępniana z kopią publikacji.

Dane SubUser

To jest dane użytkownika subskrypcji. Dane podane w subskrypcji w tym polu zostaną dołączone jako właściwość komunikatu danych MQSubUserw każdej publikacji wysłanej do tej subskrypcji.

Właściwości publikacji

W produkcie Tabela 128 na stronie 932 jest wyświetlana lista właściwości publikacji, które są udostępniane wraz z komunikatem do publikacji.

Dostęp do tych właściwości można uzyskać bezpośrednio z folderu **MQRFH2** lub pobrać za pomocą programu MQINQMP. Produkt MQINQMP akceptuje nazwę właściwości lub nazwę **MQRFH2** jako nazwę właściwości, która ma zostać zapytana.

Tabela 128. 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	Łańcuch/liczba całkowita dodana przez publikator
MQPubFormat	mqpse.Pfmt	MQTYPE_INT32	Format komunikatu: MQRFH1 MQRFH2 PCF

Porządkowanie komunikatów

W przypadku konkretnego tematu komunikaty są publikowane przez menedżer kolejek w takiej samej kolejności, w jakiej są odbierane z aplikacji publikowania (z zastrzeżeniem zmiany kolejności w oparciu o priorytet komunikatu).

Porządkowanie komunikatów zwykle oznacza, że każdy subskrybent odbiera komunikaty z określonego menedżera kolejek, dotyczące określonego tematu, od konkretnego publikatora w kolejności, w jakiej są one publikowane przez tego publikatora.

Jednak, podobnie jak w przypadku wszystkich komunikatów produktu IBM MQ , komunikaty mogą być dostarczane od czasu do czasu poza kolejną kolejką komunikatów. Sytuacja taka może wystąpić w następujących sytuacjach:

- Jeśli odsyłacz w sieci zostanie wyłączony, a kolejne komunikaty zostaną przekierowane na inny odsyłacz,
- Jeśli kolejka zostanie tymczasowo zapełniona lub zablokowana, tak aby komunikat został umieszczony w kolejce niedostarczonych komunikatów, a tym samym opóźniony, podczas gdy kolejne komunikaty są przekazywane prosto przez użytkownika.
- Jeśli administrator usunie menedżera kolejek, gdy publikatorzy i subskrybenci nadal działają, powodując umieszczenie w kolejce komunikatów, które mają zostać umieszczone w kolejce niedostarczonych komunikatów i subskrypcje, które mają zostać przerwane.

Jeśli te okoliczności nie mogą wystąpić, publikacje są zawsze dostarczane w porządku.

Uwaga: Nie można używać zgrupowanych ani segmentowanych komunikatów z publikowania/subskrybowania.

Przechwytywanie publikacji

Można przechwycić publikację, zmodyfikować ją, a następnie ponownie opublikować, zanim trafi do dowolnego innego subskrybenta.

Aby można było wykonać jedną z następujących czynności, warto przechwycić publikację, zanim zostanie on osiągnięty przez subskrybenta.

- Dołączanie dodatkowych informacji do wiadomości
- Zablokuj komunikat
- Transformowanie komunikatu

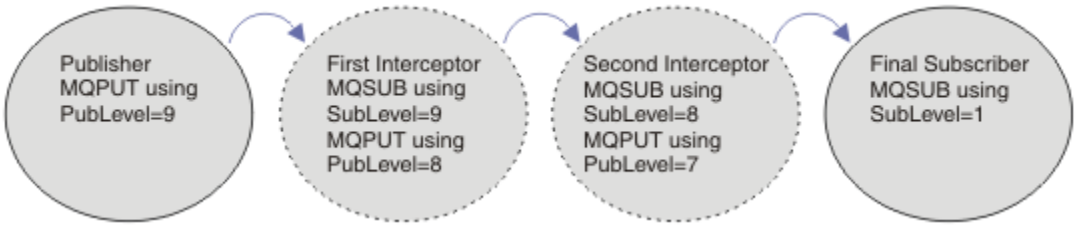
Tę samą operację można wykonać dla każdego komunikatu lub udostępnić operację w zależności od subskrypcji, komunikatu lub nagłówka komunikatu.

Odsyłacze pokrewne

MQ_PUBLISH_EXIT-wyjście publikowania

Poziomy subskrypcji

Ustaw poziom subskrypcji subskrypcji, aby przechwycić publikację, zanim zostanie osiągnięty jej finalny subskrybent. Przechwytywacz subskrybent subskrybuje wyższy poziom subskrypcji, a republiki na niższym poziomie publikacji. Budowanie łańcucha przechwytującego subskrybentów w celu przetwarzania komunikatów w publikacji przed dostarczyniem ich do końcowych subskrybentów.



Rysunek 91. Sekwencja przechwytujących subskrybentów

Aby przechwycić publikację, należy użyć atrybutu **MQSD** SubLevel . Po przechwyceniu komunikatu może on zostać przekształcony, a następnie ponownie opublikowany na niższym poziomie publikacji, zmieniając atrybut **MQPMO** PubLevel . Następnie komunikat trafia do końcowych subskrybentów lub zostanie ponownie przechwycony przez subskrybent pośredni na niższym poziomie subskrypcji.

Przechwytywacz subskrybent zwykle transformuje komunikat przed ponownym jego opublikowaniem. Sekwencja przechwytujących subskrybentów tworzy przepływ komunikatów. Alternatywnie można nie ponownie opublikować przechwyconej publikacji: Subskrybenty na niższych poziomach subskrypcji nie otrzymają komunikatu.

Upewnij się, że przechwytywacz odbiera publikacje przed innymi subskrybentami. Ustaw poziom subskrypcji przechwytywacza wyżej niż inni subskrybenci. Domyślnie subskrybenci mają SubLevel z 1. Najwyższa wartość to 9. Publikacja musi rozpoczynać się od PubLevel co najmniej tak samo wysoki, jak najwyższy SubLevel. Opublikuj początkowo z domyślnym PubLevel produktu 9.

- Jeśli w temacie został przechwycony subskrybent, ustaw wartość SubLevel na 9.
- W przypadku wielu przechwytywanych aplikacji w danym temacie należy ustawić niższą wartość SubLevel dla każdego kolejnego przechwytującego subskrybenta.
- Istnieje możliwość zaimplementowania maksymalnie 8 przechwytywanych aplikacji z poziomem subskrypcji od 9 w dół do 2 włącznie. Odbiorca końcowy komunikatu ma wartość SubLevel produktu 1.

Przechwytywacz o najwyższym poziomie subskrypcji, który jest równy lub niższy niż PubLevel publikacji, otrzymuje publikację w pierwszej kolejności. Skonfiguruj tylko jeden przechwytywacz subskrybenta dla tematu na określonym poziomie subskrypcji. Posiadanie wielu subskrybentów na określonym poziomie subskrypcji powoduje, że wiele kopii publikacji jest wysyłanych do końcowego zestawu aplikacji subskrybujących.

Subskrybent o wartości SubLevel produktu 0 jest używany jako catchall. Otrzymuje on publikację, jeśli żaden końcowy subskrybent nie otrzyma komunikatu. Subskrybent o wartości SubLevel produktu 0 może być używany do monitorowania publikacji, które nie zostały odebrane przez innych subskrybentów.

Programowanie przechwytującego subskrybenta

Użyj opcji subskrypcji opisanych w sekcji Tabela 129 na stronie 934.

Tabela 129. Opcje subskrypcji dla przechwytywaczy subskrybentów

Opcja subskrypcji	**Uwagi**
Opcje MQSO_SET_CORREL_ID i SubCorrelId ustawione na wartość MQCI_NONE	Zachowaj wartość CorrelId przechwyconej publikacji tak samo, jak oryginalna publikacja. **Uwaga:** Nie można przekazać identyfikatora korelacji publikacji w hierarchii. Pole jest używane przez menedżer kolejek.
Parametr PubPriority ustawiony na wartość MQPRI_PRIORITY_AS_PUBLISHED	Należy zachować priorytet przechwyconej publikacji, tak samo jak oryginalna publikacja.

Opcje w programie Tabela 129 na stronie 934 muszą być używane przez wszystkie przechwytujące subskrybenty. Wynika to z tego, że identyfikator korelacji i priorytet komunikatu nie są modyfikowane z ustawienia pierwotnego publikatora.

Po przetworzeniu publikacji przez subskrybent zostanie ponownie wysłany komunikat do tego samego tematu na poziomie PubLevel o wartości niższej niż SubLevel w ramach własnej subskrypcji. Jeśli przechwytywający subskrybent ustawił element SubLevel w produkcie 9, zostanie on ponownie opublikowany przez użytkownika o wartości PubLevel produktu 8.

Aby ponownie opublikować komunikat, wymagane jest kilka fragmentów informacji z oryginalnej publikacji. Ponownie wykorzystaj tę samą **MQMD** co w oryginalnym komunikacie i ustaw MQPMO_PASS_ALL_CONTEXT , aby zapewnić, że wszystkie informacje w **MQMD** zostaną przekazane do następnego subskrybenta. Skopiuj wartości z właściwości komunikatu, które są wyświetlane w Tabela

130 na stronie 935 , do odpowiednich pól ponownie opublikowanego komunikatu. Przechwytywacz subskrybenta może zmienić te wartości. Użyj operatora OR, aby dodać dodatkowe wartości do **MQPMO**. Pole `Opcje` służy do łączenia opcji umieszczania komunikatów.

Należy jawnie otworzyć kolejkę publikacji, a nie używać zarządzanej kolejki publikacji. Nie można ustawić `MQSO_SET_CORREL_ID` dla kolejki zarządzanej. Nie można również ustawić `MQOO_SAVE_ALL_CONTEXT` w kolejce zarządzanej. Zapoznaj się z fragmentami kodu wymienionymi w sekcji "Przykłady" na stronie 935.

Tabela 130. Wartości MQPUT dla ponownie opublikowanych komunikatów

Ponownie publikuj komunikat przy użyciu komendy MQPUT	**Informacje w komunikacie o publikacji**
MQOD. `ObjectString`	Właściwość komunikatu `MQTopicString`
MQPMO. `Options`	Właściwość komunikatu `MQPubOptions`

Ostatni subskrybent ma możliwość wyboru ustawienia opcji subskrypcji w inny sposób. Na przykład może ona jawnie ustawić priorytet publikowania, a nie na `MQPRI_PRIORITY_AS_PUBLISHED`. Ustawienia subskrybenta końcowego mają wpływ tylko na publikację z poziomu końcowego przechwytującego subskrybenta w łańcuchu.

Zachowane publikacje

Zachowana publikacja musi zostać zachowana po przechwyceniu, kopiując oryginalne opcje put-message do ponownie opublikowanego komunikatu.

Opcja `MQPMO_RETAIN` jest ustawiana przez publikator. Każdy przechwytywacz subskrybenta musi przesłać `MQPubOptions` do opcji put-message z ponownie opublikowanego komunikatu, jak to pokazano na Tabela 130 na stronie 935. Kopiowanie opcji put-message powoduje zachowanie opcji ustawionych przez pierwotnego publikatora, w tym także informacje o tym, czy ma być zachowana publikacja.

Gdy publikacja kończy swój fragment łańcucha przechwytującego subskrybentów, i jest dostarczany do końcowych subskrybentów, to w końcu zostaje zachowana. Nowi subskrybenci, na poziomie `SubLevel` 1, żądający zachowanej publikacji, otrzymują ją bez żadnych dodatkowych przechwytów. Subskrybenty `SubLevel` większe niż 1 nie są wysyłane do zachowanej publikacji. W związku z tym zachowana publikacja nie jest modyfikowana przez łańcuch przechwytywania subskrybentów po raz drugi przez cały czas.

Przykłady

Przykłady to fragmenty kodu, które można połączyć w celu zbudowania przechwytywacza. Kod jest zapisywany jako krótki, a nie jako jakość produkcji.

Dyrektywy preprocesora w produkcie Rysunek 92 na stronie 936 definiują dwie właściwości, które mają zostać wyodrębnione z komunikatów publikacji, które są wymagane 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 92. Dyrektywy preprocesora

Rysunek 93 na stronie 936 zawiera listę deklaracji używanych w fragmentach kodu. Deklaracje są standardowe dla aplikacji IBM MQ , z wyjątkiem podświetlonych terminów.

Podświetlone opcje umieszczania i pobierania są inicjowane w celu przekazania całego kontekstu. Podświetlone pola MQTOPICSTRING i MQPUBOPTIONS to inicjatory MQCHARV dla nazw właściwości, które są zdefiniowane w dyrektywach preprocesora. Nazwy są przekazywane do produktu 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 93. Deklaracje

Inicjacje, które nie są łatwo wykonywane w deklaracjach, są wyświetlane w produkcie Rysunek 94 na stronie 937. Podświetlone wartości wymagają wyjaśnienia.

SYSTEM.NDURABLE.MODEL.QUEUE
W tym przykładzie zamiast korzystania z produktu MQSUB w celu otwarcia zarządzanej nietrwałej subskrypcji, kolejka modelowa SYSTEM.NDURABLE.MODEL.QUEUEjest używana do tworzenia tymczasowej kolejki dynamicznej. Jego uchwyt jest przekazywany do produktu MQSUB. Bezpośrednio

otwierając kolejkę, można zapisać cały kontekst komunikatu i ustawić opcję subskrypcji, MQSO_SET_CORREL_ID.

MQGMO_CURRENT_VERSION
Ważne jest, aby używać bieżącej wersji większości struktur produktu IBM MQ . Pola, takie jak gmo.MsgHandle , są dostępne tylko w najnowszej wersji struktur sterujących.

MQGMO_PROPERTIES_IN_HANDLE
Łańcuch tematu i opcje umieszczania komunikatów ustawione w oryginalnej publikacji mają być pobierane przez przechwytywanego subskrybenta przy użyciu właściwości komunikatu. Alternatywnym rozwiązaniem jest bezpośrednie odczytanie 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 pierwotny publikator jest umieszczany w polu identyfikatora korelacji publikacji odebranej przez przechwytywanego subskrybenta. Każdy przechwytywany subskrybent przechodzi na ten sam identyfikator korelacji. Następnie końcowy subskrybent ma opcję otrzymywania tego samego identyfikatora korelacji.

Uwaga: Jeśli publikacja jest przekazywana za pośrednictwem hierarchii publikowania/subskrypcji, identyfikator korelacji nigdy nie jest zachowywany.

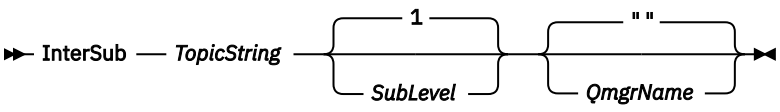
MQPRI_PRIORITY_AS_PUBLISHED
Publikacja jest umieszczana w kolejce publikacji z tym samym priorytetem komunikatów, w którym został opublikowany.

```
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 94. Inicjalizacje

Rysunek 95 na stronie 938 przedstawia fragment kodu w celu odczytania parametrów wiersza komend, zakończenie inicjowania i utworzenie subskrypcji przechwytującej.

Uruchom program z komendą,



Aby obsługa błędów była niezauważalna, kod przyczyny z każdego wywołania MQI jest przechowywany w innym elemencie tablicy. Po każdym wywołaniu kod zakończenia jest testowany, a jeśli wartością jest MQCC_FAIL, element sterujący powoduje wyjście z bloku kodu do `{ } while(0)` .

Dwie warte uwagi linie kodu to:

pmo.PubLevel = sd.SubLevel - 1;
Ustawia poziom publikacji dla ponownie opublikowanego komunikatu na jeden mniejszy niż poziom subskrypcji przechwytującego subskrybenta.

gmo.MsgHandle = Hmsg;
Udostępnia uchwyt komunikatu dla produktu 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 95. Przygotowanie do przechwycenia publikacji

Główny fragment kodu, Rysunek 96 na stronie 939, pobiera komunikaty z kolejki publikacji. Wysyła on zapytania do właściwości komunikatu i ponownie publikuje komunikaty przy użyciu łańcucha tematu i oryginalnego **MQPMO**. Właściwości opcji publikacji.

W tym przykładzie w publikacji nie jest wykonywana żadna transformacja. Łańcuch tematu ponownie opublikowanej publikacji jest zawsze zgodny z łańcuchem tematu przechwytującą subskrybent subskrybowany. Jeśli przechwytywacz subskrybenta jest odpowiedzialny za przechwytywanie wielu subskrypcji wysłanych do tej samej kolejki publikacji, może być konieczne wysłanie zapytania do łańcucha tematu w celu odróżnienia publikacji, które są zgodne z różnymi subskrypcjami.

Wywołania programu MQINQMP są podświetlone. Łańcuch tematu i publikacja właściwości opcji umieszczania komunikatów są zapisywane bezpośrednio w strukturach sterujących wyjścia. Jedynym powodem zmiany pola długości MQCHARV putOD.ObjectString z jawnej długości do łańcucha zakończonego znakiem NULL jest użycie 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 96. Przechwyć publikację i ponownie opublikuj

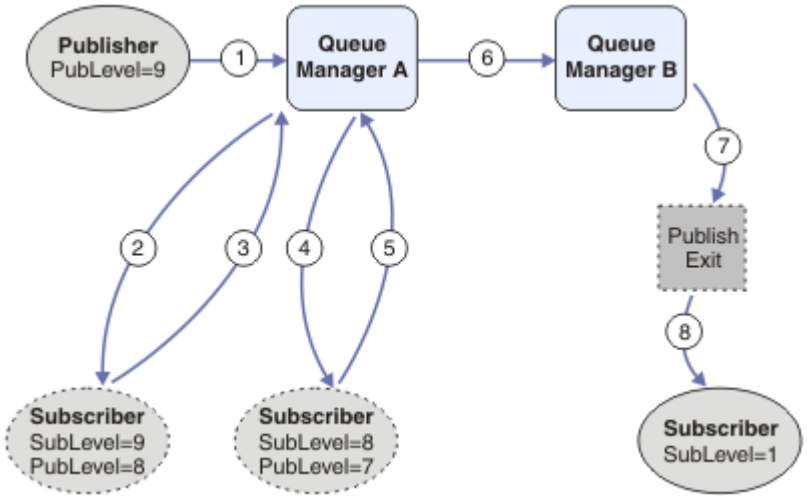
Końcowy fragment kodu jest wyświetlany w programie Rysunek 97 na stronie 939.

```
    } 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 97. Zakończenie

Przechwytywanie publikacji i rozproszonego publikowania/subskrypcji
Po wdrożeniu przechwytywanych subskrybentów lub publikowania wyjść do rozproszonej topologii publikowania/subskrypcji należy zastosować prosty wzorzec. Wdróż przechwytywanie subskrybentów w tych samych menedżerach kolejek, co publikatory, i opublikuj wyjścia w tych samych menedżerach kolejek co końcowe subskrybenci.

W produkcie Rysunek 98 na stronie 940 są wyświetlane dwa menedżery kolejek połączone w klastrze publikowaniusubskrypcji. Publikator tworzy publikację do tematu klastra na poziomie publikacji 9. Strzałki numerowane przedstawiają sekwencję kroków podjętych przez publikację w miarę przepływów do subskrybentów tematu klastra. Publikacja jest przechwytywana przez subskrybenta za pomocą elementu `Sublevel` 9 i ponownie publikowana za pomocą programu `Publevel` 8. Jest on ponownie przechwytywany przez subskrybenta na poziomie `Podpoziom` 8. Subskrybent zostanie ponownie opublikowany na poziomie `Publevel` 7. Subskrybent proxy udostępniony przez menedżer kolejek przekazuje publikację do menedżera kolejek B, w którym oprócz subskrybenta końcowego wdrożono wyjście publikowania. Publikacja jest przetwarzana przez wyjście publikowania, zanim zostanie ostatecznie odebrana przez końcowego subskrybenta na poziomie `Podpoziom` 1. Przechwytywani subskrybenci i wyjście publikowania są wyświetlane z niepoprawnymi konturami.



Rysunek 98. Wyjście przechwycenia i publikowania w klastrze

Celem prostego wzorca jest dla każdego abonenta, który otrzymuje publikację, aby otrzymać identyczną publikację. Publikacja przechodzi przez tę samą sekwencję transformacji niezależnie od miejsca, w którym subskrybent jest połączony. Prawdopodobnie chcesz uniknąć zmiany sekwencji transformacji, w zależności od tego, gdzie są połączone publikatory lub finałowe subskrybenci. Rozsądnym wyjątkiem byłoby dostosowanie publikacji w końcu dostarczonej do każdego pojedynczego abonenta. Użyj wyjścia publikowania, aby dostosować publikację w oparciu o kolejkę, do której ostatecznie została dostarczona publikacja.

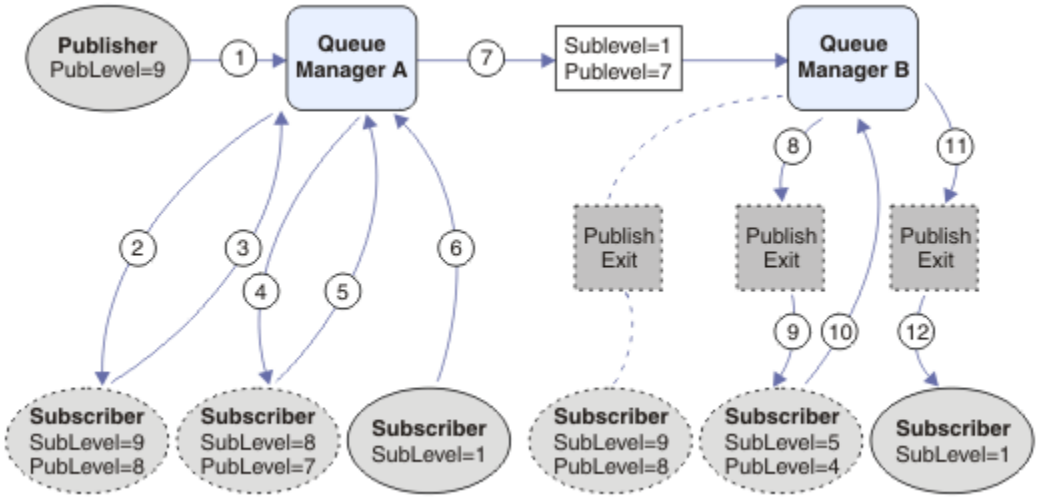
Należy dokładnie rozważyć miejsce wdrożenia przechwytywanych subskrybentów i wyjść publikowania w rozproszonej topologii publikowania/subskrypcji. Wzorzec prosty wdraża przechwytywanie subskrybentów do tego samego menedżera kolejek co publikatory, a publikowanie kończy się w tych samych menedżerach kolejek, co w przypadku końcowych subskrybentów.

Anty-wzorzec

Rysunek 99 na stronie 941 pokazuje, w jaki sposób można wykonywać awry, jeśli nie jest zgodny z prostym wzorcem. Aby skomplikować wdrożenie, do menedżera kolejek A dodawany jest ostatni subskrybent, a do menedżera kolejek B dodawane są dwa dodatkowe subskrybenty przechwytujące.

Publikacja jest przekazywana do menedżera kolejek B na poziomie `PubLevel` 7, gdzie jest przechwytywana przez subskrybenta na poziomie `SubLevel` 5 przed ich zużyciu przez końcowego subskrybenta na poziomie `SubLevel` 1. Wyjście publikowania przechwytuje publikację, zanim zostanie przekazana zarówno do przechwytującego konsumenta, jak i do konsumenta końcowego w menedżerze kolejek B. Publikacja dociera do końcowego subskrybenta menedżera kolejek A bez przetwarzania przez wyjście publikowania.

W topologii publikowania/subskrypcji subskrybenci proxy subskrybują element `SubLevel` 1i przekazują wartości `PubLevel` ustawione przez ostatni przechwytywacz subskrybenta. W produkcie Rysunek 99 na stronie 941wynik jest taki, że subskrybent nie przechwycił publikacji przy użyciu elementu `SubLevel` 9 w menedżerze kolejek B.



Rysunek 99. Złożone wdrożenie przechwytywanych subskrybentów

Opcje publikowania

Dostępnych jest kilka opcji, które sterują sposobem publikowania komunikatów.

Wstrzymać odpowiedź-do informacji od subskrybentów

Jeśli nie chcesz, aby subskrybenci mieli możliwość odpowiadania na otrzymane publikacje, możliwe jest wstrzymanie informacji w polach ReplyToQ i ReplyToQmgr w strukturze MQMD za pomocą opcji put-message w tabeli MQPMO_SUPPRESS_REPLYTO. Jeśli ta opcja jest używana, menedżer kolejek usuwa te informacje z deskryptora MQMD, gdy otrzymuje on publikację przed przekazaniem go do wszystkich subskrybentów.

Tej opcji nie można używać w połączeniu z opcją raportu, która wymaga kolejki ReplyTo, jeśli próba ta jest podejmowana z powodu niepowodzenia wywołania MQRC_MISSING_REPLY_TO_Q.

Poziom publikacji

Korzystanie z poziomów publikacji jest sposobem na kontrolowanie, którzy subskrybenci otrzymują publikację. Poziom publikacji oznacza poziom subskrypcji, do którego odnosi się publikacja. Publikacja otrzyma tylko subskrypcje o najwyższym poziomie subskrypcji, mniejsze lub równe poziomowi publikacji publikacji. Wartość ta musi być z 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 żadnych subskrybentów

Aby sprawdzić, czy publikacja nie została dostarczona do żadnych subskrybentów, należy użyć opcji put-message komendy MQPMO_WARN_IF_NO_SUBS_MATCHED z wywołaniem MQPUT. Jeśli kod zakończenia komendy MQCC_WARNING i kod przyczyny MQRC_NO_SUBS_MATCHED zostaną zwrócone przez operację put, publikacja nie została dostarczona do żadnych subskrypcji. Jeśli w operacji put została określona opcja MQPMO_RETAIN, komunikat jest zachowywany i dostarczany do dowolnej późniejszej definicji zgodnej subskrypcji. W rozproszonym systemie publikowania/subskrybowania kod przyczyny MQRC_NO_SUBS_MATCHED jest zwracany tylko wtedy, gdy dla tematu w menedżerze kolejek nie zarejestrowano żadnych subskrypcji proxy.

Opcje subskrypcji

Dostępnych jest kilka opcji, które sterują sposobem obsługi subskrypcji komunikatów.

Trwałość komunikatu

Menedżery kolejek zachowują trwałość publikacji, które przesyłają do subskrybentów, zgodnie z ustawionym przez publikatora. Publikator ustawia trwałość, aby była jedną z następujących opcji:

0
Nietrwałe

1
Trwałe

2
Trwałość jako definicja kolejki/tematu

W przypadku publikowania/subskrypcji publikator rozstrzyża obiekt tematu i obiekt **topicString** na rozstrzygnięty obiekt tematu. Jeśli publikator określa trwałość jako definicję kolejki/tematu, to domyślna trwałość z rozstrzygniętego obiektu tematu jest ustawiana na potrzeby publikacji.

Zachowane publikacje

Aby kontrolować czas odbierania zachowanych publikacji, subskrybenci mogą korzystać z dwóch opcji subskrypcji:

Publikuj tylko na żądanie, MQSO_PUBLICATIONS_ON_REQUEST

Jeśli subskrybent ma mieć kontrolę nad publikacjami, można skorzystać z opcji subskrypcji MQSO_PUBLICATIONS_ON_REQUEST. Subskrybent może sterować tym, kiedy otrzymuje publikacje za pomocą wywołania MQSUBRQ (określając uchwyt Hsub, który został zwrócony z oryginalnego wywołania MQSUB), aby zażądać, aby został wysłany zachowaną publikację tematu. Subskrybenci korzystający z opcji subskrypcji MQSO_PUBLICATIONS_ON_REQUEST nie otrzymują żadnych niezachowanych publikacji.

Jeśli zostanie określona wartość MQSO_PUBLICATIONS_ON_REQUEST, należy użyć komendy MQSUBRQ w celu pobrania dowolnej publikacji. Jeśli nie zostanie użyta wartość MQSO_PUBLICATIONS_ON_REQUEST, zostaną wyświetlone komunikaty w postaci, w której są one publikowane.

Jeśli subskrybent korzysta z wywołania MQSUBRQ i korzysta ze znaków wieloznacznych w temacie subskrypcji, subskrypcja może być zgodna z wieloma tematami lub węzłami w drzewie tematów, a wszystkie z zachowanych komunikatów (o ile istnieją) zostaną wysłane do subskrybenta.

Ta opcja może być szczególnie przydatna w przypadku użycia z trwałymi subskrypcjami, ponieważ menedżer kolejek będzie nadal wysyłać publikacje do subskrybenta, jeśli subskrybent jest subskrybowany nawet wtedy, gdy ta aplikacja subskrybenta nie jest uruchomiona. Może to prowadzić do budowania komunikatów w kolejce subskrybenta. Tej kompilacji można uniknąć, jeśli subskrybent zarejestruje się za pomocą opcji MQSO_PUBLICATIONS_ON_REQUEST. Alternatywnie można użyć nietrwałych subskrypcji, jeśli jest to właściwe dla aplikacji, aby uniknąć tworzenia niechcianych komunikatów.

Jeśli subskrypcja jest trwała, a publikator korzysta z zachowanych publikacji, aplikacja subskrybenta może użyć wywołania MQSUBRQ w celu odświeżenia informacji o stanie po restarcie. Subskrybent musi następnie okresowo odświeżać swój stan przy użyciu wywołania MQSUBRQ.

Żadne publikacje nie będą wysyłane w wyniku wywołania MQSUB za pomocą tej opcji. Trwała subskrypcja, która została wznowiona po rozłączonym połączeniu, będzie używała opcji MQSO_PUBLICATIONS_ON_REQUEST, jeśli pierwotna subskrypcja została skonfigurowana w taki sposób, aby korzystała z tej opcji.

Tylko nowe publikacje, MQSO_NEW_PUBLICATIONS_ONLY

Jeśli zachowana publikacja istnieje w temacie, wszyscy subskrybenci, którzy dokonają subskrypcji po publikacji, otrzymają kopię tej publikacji. Subskrybent może korzystać z opcji subskrypcji MQSO_NEW_PUBLICATIONS_ONLY, jeśli subskrybent nie chce otrzymywać żadnych publikacji, które zostały wykonane wcześniej niż subskrypcja.

Grupowanie subskrypcji

Subskrypcje grupujące należy rozważyć, jeśli kolejka została ustawiona w celu odbierania publikacji i istnieje pewna liczba nakładających się na siebie subskrypcji, które publikują publikacje w tej samej kolejce. Ta sytuacja jest podobna do sytuacji w sekcji Overlapping subskrypcji.

Aby uniknąć otrzymywania zduplikowanych publikacji, należy ustawić opcję MQSO_GROUP_SUB w momencie subskrybowania tematu. W wyniku tego, gdy 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 zgodny łańcuch tematu przed napotkaniem wszystkich znaków wieloznacznych. Może być on pomyślany jako najbliższa zgodna subskrypcja. Jego właściwości są propagowane do publikacji, w tym, bez względu na to, czy ma właściwość MQSO_NOT_OWN_PUBS . Jeśli tak, żadna publikacja nie zostanie dostarczona do kolejki, mimo że inne zgodne subskrypcje mogą nie mieć właściwości MQSO_NOT_OWN_PUBS .

Nie można umieścić wszystkich subskrypcji w jednej grupie w celu wyeliminowania zduplikowanych publikacji. Pogrupowane 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 być na tym samym poziomie subskrypcji.
4. Komunikat publikacji dla każdej subskrypcji w grupie ma taki sam identyfikator korelacji.

   Aby upewnić się, że każda subskrypcja powoduje utworzenie komunikatu publikacji o tym samym identyfikatorze korelacji, należy ustawić wartość MQSO_SET_CORREL_ID , aby utworzyć własny identyfikator korelacji w publikacji, a następnie 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 ../com.ibm.mq.ref.dev.doc/q100080_.dita#q100080_/mqso_group_sub .

Sprawdzanie i ustawianie atrybutów obiektu

Atrybuty to właściwości, które definiują parametry obiektu IBM MQ .

Wpływają one na sposób, w jaki menedżer kolejek przetwarza obiekt. Atrybuty każdego typu obiektu IBM MQ są szczegółowo opisane w sekcji Atrybuty obiektów.

Niektóre atrybuty są ustawiane przy definiowaniu 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, które mogą zmieniać się w czasie. Przykładem może być bieżąca głębokość kolejki.

Za pomocą wywołania MQINQ można zapytać o bieżące wartości większości atrybutów. Interfejs MQI udostępnia także wywołanie MQSET, z którym można zmienić niektóre atrybuty kolejki. Nie można używać wywołań MQI w celu zmiany atrybutów dowolnego innego typu obiektu. Zamiast tego należy użyć jednego z następujących zasobów:

- ULW Narzędzie MQSC, które jest opisane w sekcji Komendy MQSC.
- IBM i Komendy CL CHGMQMx, które zostały opisane w sekcji Skorowidz komend CL dla produktu IBM ilub w narzędziu MQSC.
- z/OS Komendy ALTER dla operatora 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 której są używane z wywołaniami MQINQ i MQSET. Jeśli do definiowania, modyfikowania lub wyświetlania atrybutów używane są komendy produktu IBM MQ , należy zidentyfikować atrybuty za pomocą słów kluczowych przedstawionych w opisach komend w odsyłaczkach do tematów.

Zarówno wywołania MQINQ, jak i MQSET używają tablic selektorów do identyfikowania tych atrybutów, które mają zostać zapytane lub ustawione. Dla każdego atrybutu, z którym można pracować, istnieje selektor. Nazwa selektora ma przedrostek określony przez rodzaj atrybutu:

Tabela 131. Przedrostki nazw selektorów

Przedrostek	**Opis**
MQCA_	Te selektory odwołują się do atrybutów zawierających dane znakowe (na przykład nazwę kolejki).
MQIA_	Te selektory odwołują się do atrybutów, które zawierają wartości liczbowe (takie jak *CurrentQueueDepth*, liczba komunikatów w kolejce) lub wartość stała (na przykład *SyncPoint*, niezależnie od tego, czy menedżer kolejek obsługuje punkty synchronizacji).

Przed użyciem wywołania MQINQ lub MQSET należy połączyć aplikację z menedżerem kolejek i użyć wywołania MQOPEN, aby otworzyć obiekt do ustawienia lub zapytania o atrybuty. Operacje te są opisane w produktach "Nawiąże połączenie z menedżerem kolejek i odłączenie go od niego" na stronie 824 i "Otwieranie i zamykanie obiektów" na stronie 833.

Użyj poniższych odsyłaczy, aby dowiedzieć się więcej na temat uzyskiwania informacji na temat atrybutów obiektu i ustawiania atrybutów obiektu:

- "Uzyskiwanie informacji o atrybutach obiektu" na stronie 945
- "Niektóre przypadki, w których wywołanie MQINQ nie powiodło się" na stronie 945
- "Ustawianie atrybutów kolejki" na stronie 946

Pojęcia pokrewne

"Interfejs kolejki komunikatów-przegląd" na stronie 811
Informacje na temat komponentów interfejsu kolejek komunikatów (Message Queue Interface-MQI).

"Nawiąże połączenie z menedżerem kolejek i odłączenie go od niego" na stronie 824
Aby można było korzystać z usług programistycznych produktu IBM MQ , program musi mieć połączenie z menedżerem kolejek. Informacje zawarte w tej sekcji umożliwiają zapoznanie się z informacjami na temat nawiązywania połączenia z menedżerem kolejek i z jego rozłączeniem.

"Otwieranie i zamykanie obiektów" na stronie 833
Te informacje udostępniają wgląd w otwieranie i zamykanie obiektów produktu IBM MQ .

"Umieszczanie komunikatów w kolejce" na stronie 844
Te informacje umożliwiają zapoznanie się z informacjami na temat umieszczania komunikatów w kolejce.

"Pobieranie komunikatów z kolejki" na stronie 860
Ta sekcja zawiera informacje na temat pobierania komunikatów z kolejki.

"Zatwierdzanie i wycofywanie jednostek pracy" na stronie 946
W tej sekcji opisano sposób zatwierdzania i tworzenia kopii zapasowych wszystkich możliwych do odtworzenia operacji get i put, które wystąpiły w jednostce pracy.

"Uruchamianie aplikacji produktu IBM MQ przy użyciu wyzwalaczy" na stronie 958
Informacje na temat wyzwalaczy i sposobów uruchamiania aplikacji IBM MQ przy użyciu wyzwalaczy.

"Praca z interfejsem MQI i klastrami" na stronie 978
Istnieją specjalne opcje dotyczące wywołań i kodów powrotu, które odnoszą się do technologii klastrowej.

"Używanie i zapisywanie aplikacji w systemie IBM MQ for z/OS" na stronie 983
Aplikacje produktu IBM MQ for z/OS mogą być uruchamiane z programów działających w wielu różnych środowiskach. Oznacza to, że mogą skorzystać z udogodnień dostępnych w więcej niż jednym środowisku.

"Aplikacje pomostowe IMS i IMS w systemie IBM MQ for z/OS" na stronie 70
Te informacje ułatwiają pisanie aplikacji produktu IMS przy użyciu produktu IBM MQ.

Uzyskiwanie informacji o atrybutach obiektu

Użyj wywołania MQINQ, aby dowiedzieć się więcej 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 o wartości, o którą użytkownik chce się dowiedzieć, a każdy selektor musi być poprawny dla typu obiektu reprezentowanego przez uchwyt obiektu. Selektory można określać w dowolnej kolejności.
- Liczba atrybutów całkowitoliczbowych, o których pytasz. Podaj wartość zero, jeśli nie jesteś pytany o atrybuty całkowitoliczbowe.
- Długość buforu atrybutów znakowych w programie *CharAttrLength*. Musi to być co najmniej suma długości wymaganych do przechowywania każdego łańcucha atrybutu znakowego. Wartość zero oznacza, że nie jest pytany o atrybuty znaków.

Dane wyjściowe komendy MQINQ są następujące:

- Zestaw wartości atrybutów całkowitoliczbowych skopiowanych do tablicy. Liczba wartości jest określana przez produkt *IntAttrCount*. Jeśli parametr *IntAttrCount* lub *SelectorCount* ma wartość zero, ten parametr nie jest używany.
- Bufor, w którym zwracane są atrybuty znakowe. Długość buforu jest nadawana przez parametr **CharAttrLength** . Jeśli parametr *CharAttrLength* lub *SelectorCount* ma wartość zero, ten parametr nie jest używany.
- Kod zakończenia. Jeśli kod zakończenia generuje 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 całkowitych.
  - Brak wystarczającej ilości miejsca dla atrybutów znakowych

  Jeśli wystąpi więcej niż jedna z tych sytuacji, zwracana jest pierwsza z nich.

Jeśli otwarto kolejkę dla danych wyjściowych lub zapytania i zostanie ona rozstrzygana w nielokalnej kolejce klastra, można jedynie uzyskać informacje o nazwie kolejki, typie kolejki i wspólnych atrybutach. Jeśli użyto komendy MQOO_BIND_ON_OPEN, wartości wspólnych atrybutów są wartościami z wybranej kolejki. Wartości te należą do dowolnej z możliwych kolejek klastra, jeśli użyto wartości MQOO_BIND_NOT_FIXED lub MQOO_BIND_ON_GROUP, albo użyto komendy MQOO_BIND_AS_Q_DEF, a atrybut kolejki produktu **DefBind** miał wartość MQBND_BIND_NOT_FIXED. Więcej informacji na ten temat można znaleźć w sekcji “MQOPEN i klastry” na stronie 979 i MQOPEN .

Uwaga: Wartości zwracane przez wywołanie są obrazem stanu wybranych atrybutów. Atrybuty mogą się zmieniać, zanim program będzie działać na zwróconych wartościach.

W tabeli MQINQznajduje się opis wywołania MQINQ.

Niektóre przypadki, w których wywołanie MQINQ nie powiodło się

Jeśli alias zostanie otwarty w celu sprawdzenia jego atrybutów, zostaną zwrócone atrybuty kolejki aliasowej (obiekt IBM MQ używany do uzyskania dostępu do innej kolejki), a nie atrybuty kolejki podstawowej.

Jednak definicja kolejki podstawowej, do której jest rozstrzygany alias, jest również otwierana przez menedżer kolejek, a jeśli inny program zmieni użycie kolejki podstawowej w przedziale 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 w przypadku otwarcia kolejki zdalnej w celu sprawdzenia jej atrybutów zwracane są tylko atrybuty lokalnej definicji kolejki zdalnej.

W przypadku określenia jednego lub większej liczby selektorów, które nie są poprawne dla typu atrybutów kolejki, dla których jest wykonywane zapytanie, wywołanie MQINQ kończy się z ostrzeżeniem i ustawia dane wyjściowe w następujący sposób:

- W przypadku atrybutów będących liczbami całkowitymi, odpowiadające im elementy *IntAttrs* są ustawione na wartość MQIAV_NOT_APPLICABLE.
- W przypadku atrybutów znakowych odpowiednie części łańcucha *CharAttrs* są ustawiane na gwiazdki.

Jeśli zostanie określony jeden lub większa liczba selektorów, które nie są poprawne dla typu atrybutów obiektu, dla których zostanie zapytany, wywołanie MQINQ nie powiedzie się i zostanie zwrócony kod przyczyny MQRC_SELECTOR_ERROR.

Nie można wywołać funkcji MQINQ w celu wyszukania kolejki modelowej. W tym celu należy użyć narzędzia MQSC lub komend dostępnych na platformie.

Ustawianie atrybutów kolejki

Informacje zawarte w tej sekcji umożliwiają poznanie sposobu ustawiania atrybutów kolejki przy użyciu 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 dla tabeli MQSET wszystkie parametry z wyjątkiem kodu zakończenia i kodu przyczyny są parametrami wejściowymi. Brak sytuacji częściowych.

Uwaga: Nie można użyć interfejsu MQI w celu ustawienia 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 tworzenia kopii zapasowych wszystkich możliwych do odtworzenia operacji get i put, które wystąpiły w jednostce pracy.

W tym temacie są używane następujące terminy:

- Zatwierdzanie
- Wycofanie
- Koordynacja punktu synchronizacji
- Punkt synchronizacji
- Jednostka pracy
- zatwierdzanie jednofazowe
- zatwierdzania dwufazowe.

Użytkownik zaznajomiony z tymi terminami przetwarzania transakcji może przejść do programu "Uwagi dotyczące punktu synchronizacji w aplikacjach produktu IBM MQ" na stronie 948.

Zatwierdzanie i wycofanie
Gdy program umieszcza komunikat w kolejce w obrębie jednostki pracy, ten komunikat jest widoczny dla innych programów tylko wtedy, gdy program zatwierdza jednostkę pracy. Aby zatwierdzić jednostkę pracy, wszystkie aktualizacje muszą być pomyślne, aby zachować integralność danych. Jeśli program wykryje błąd i zdecyduje, że operacja put nie jest trwała, może wycofać się z jednostki pracy. Gdy program wykonuje wycofany program, program IBM MQ odtwarza kolejkę, usuwając komunikaty umieszczone w kolejce przez tę jednostkę pracy. Sposób, w jaki program wykonuje operacje zatwierdzania i tworzenia kopii zapasowych, zależy od środowiska, w którym działa program.

Podobnie, gdy program pobiera komunikat z kolejki w obrębie jednostki pracy, komunikat ten pozostaje w kolejce do momentu zatwierdzenia przez program jednostki pracy, ale komunikat nie jest dostępny do pobrania przez inne programy. Komunikat zostaje trwale usunięty z kolejki, gdy program zatwierdza jednostkę pracy. Jeśli program tworzy kopię zapasową jednostki pracy, program IBM MQ odtwarza kolejkę, udostępniając komunikaty do pobrania przez inne programy.

Koordynacja punktu synchronizacji, punkt synchronizacji, jednostka pracy
Koordynacja punktu synchronizacji to proces, za pomocą którego jednostki pracy są zatwierdzane lub wycofane z integralności danych.

Decyzja o zatwierdzeniu lub wycofaniu zmian jest podejmowana, w najprostszym przypadku, na końcu transakcji. Może być jednak bardziej przydatne dla aplikacji w celu zsynchronizowania zmian danych w innych punktach logicznych w ramach transakcji. Te punkty logiczne są nazywane *punktami synchronizacji* (lub *punktami synchronizacji*). i okres przetwarzania zbioru aktualizacji między dwoma punktami synchronizacji jest nazywany *jednostką pracy*. Kilka wywołań MQGET i wywołań MQPUT może być częścią pojedynczej jednostki pracy.

Maksymalna liczba komunikatów w jednostce pracy może być sterowana przez atrybut MAXUMSGS komendy ALTER QMGR .

zatwierdzanie jednofazowe
Proces *zatwierdzania jednofazowego* to proces, w którym program może zatwierdzać aktualizacje w kolejce bez koordynacji jej zmian z innymi menedżerami zasobów.

zatwierdzania dwufazowe.
Proces *zatwierdzania dwufazowego* to proces, w którym aktualizacje, które program wykonał dla kolejek produktu IBM MQ , mogą być koordynowane z aktualizacjami innych zasobów (na przykład baz danych pod kontrolą produktu Db2). W ramach takiego procesu aktualizacje wszystkich zasobów są zatwierdzane lub wycofane razem.

Aby ułatwić obsługę jednostek pracy, produkt IBM MQ udostępnia atrybut **BackoutCount** . Wartość ta jest zwiększana za każdym razem, gdy zostanie utworzona kopia zapasowa komunikatu w ramach jednostki pracy. Jeśli komunikat wielokrotnie powoduje nieprawidłowe zakończenie działania jednostki pracy, wartość *BackoutCount* w końcu przekracza wartość *BackoutThreshold*. Ta wartość jest ustawiana, gdy kolejka jest zdefiniowana. W takiej sytuacji aplikacja może usunąć komunikat z jednostki pracy i umieścić ją w innej kolejce, zgodnie z definicją w sekcji *BackoutRequeueQName*. Gdy komunikat jest przenoszony, jednostka pracy może zatwierdzić.

Aby dowiedzieć się więcej na temat zatwierdzania i tworzenia kopii zapasowych jednostek pracy, należy użyć następujących odsyłaczy:

- "Uwagi dotyczące punktu synchronizacji w aplikacjach produktu IBM MQ" na stronie 948
- z/OS "Punkty synchronizacji w aplikacjach produktu IBM MQ for z/OS" na stronie 950
- IBM i "Punkty synchronizacji w produkcie CICS dla aplikacji IBM i" na stronie 952
- "Punkty synchronizacji w produkcie IBM MQ for Multiplatforms" na stronie 952
- IBM i "Interfejsy do zewnętrznego menedżera synchronizacji punktu synchronizacji produktu IBM i" na stronie 957

Pojęcia pokrewne

“Interfejs kolejki komunikatów-przegląd” na stronie 811
Informacje na temat komponentów interfejsu kolejek komunikatów (Message Queue Interface-MQI).

“Nawiąże połączenie z menedżerem kolejek i odłączenie go od niego” na stronie 824
Aby można było korzystać z usług programistycznych produktu IBM MQ , program musi mieć połączenie z menedżerem kolejek. Informacje zawarte w tej sekcji umożliwiają zapoznanie się z informacjami na temat nawiązywania połączenia z menedżerem kolejek i z jego rozłączeniem.

“Otwieranie i zamykanie obiektów” na stronie 833
Te informacje udostępniają wgląd w otwieranie i zamykanie obiektów produktu IBM MQ .

“Umieszczanie komunikatów w kolejce” na stronie 844
Te informacje umożliwiają zapoznanie się z informacjami na temat umieszczania komunikatów w kolejce.

“Pobieranie komunikatów z kolejki” na stronie 860
Ta sekcja zawiera informacje na temat pobierania komunikatów z kolejki.

“Sprawdzanie i ustawianie atrybutów obiektu” na stronie 943
Atrybuty to właściwości, które definiują parametry obiektu IBM MQ .

“Uruchamianie aplikacji produktu IBM MQ przy użyciu wyzwalaczy” na stronie 958
Informacje na temat wyzwalaczy i sposobów uruchamiania aplikacji IBM MQ przy użyciu wyzwalaczy.

“Praca z interfejsem MQI i klastrami” na stronie 978
Istnieją specjalne opcje dotyczące wywołań i kodów powrotu, które odnoszą się do technologii klastrowej.

“Używanie i zapisywanie aplikacji w systemie IBM MQ for z/OS” na stronie 983
Aplikacje produktu IBM MQ for z/OS mogą być uruchamiane z programów działających w wielu różnych środowiskach. Oznacza to, że mogą skorzystać z udogodnień dostępnych w więcej niż jednym środowisku.

“Aplikacje pomostowe IMS i IMS w systemie IBM MQ for z/OS” na stronie 70
Te informacje ułatwiają pisanie aplikacji produktu IMS przy użyciu produktu IBM MQ.

Uwagi dotyczące punktu synchronizacji w aplikacjach produktu IBM MQ

Ta sekcja zawiera informacje na temat używania punktów synchronizacji w aplikacjach produktu IBM MQ .

Zatwierdzanie dwufazowe jest obsługiwane przez następujące środowiska:

- AIX IBM MQ for AIX
- IBM i IBM MQ for IBM i
- Linux IBM MQ for Linux
- Solaris IBM MQ for Solaris
- Windows IBM MQ for Windows
- z/OS CICS Transaction Server for z/OS
- z/OS TXSeries
- z/OS IMS/ESA
- z/OS z/OS -zadanie wsadowe z RRS
- Inni koordynatorzy zewnętrzni korzystający z interfejsu XA X/Open

Zatwierdzanie jednofazowe jest obsługiwane przez następujące środowiska:

- IBM i IBM MQ for IBM i
- UNIX IBM MQWŁĄCZONE UNIX
- Windows IBM MQ for Windows

- z/OS z/OS wsadowe

Więcej informacji na temat interfejsów zewnętrznych zawiera sekcja “Interfejsy do zewnętrznych menedżerów punktów synchronizacji w wielu platformach” na stronie 955oraz dokumentacja interfejsu XA *CAE Specification Distributed Transaction Processing: The XA Specification*(Specyfikacja rozproszonego przetwarzania transakcji CAE: Specyfikacja XA) publikowana przez grupę Open Group. Menedżery transakcji (takie jak CICS, IMS, Encina i Tuxedo) mogą uczestniczyć w zatwierdzaniu dwufazowym, które są koordynowane z innymi zasobami odtwarzalnymi. Oznacza to, że funkcje kolejkowania udostępniane przez produkt IBM MQ mogą być wprowadzane w zasięgu jednostki pracy, zarządzanej przez menedżera transakcji.

Przykłady dostarczone wraz z programem IBM MQ pokazują IBM MQ koordynujące bazy danych zgodne z protokołem XA. Więcej informacji na temat tych przykładów można znaleźć w sekcji “Korzystanie z przykładowych programów proceduralnych produktu IBM MQ” na stronie 1165.

W aplikacji IBM MQ można określić w każdym wywołaniu i uzyskać informacje o tym, czy wywołanie ma być pod kontrolą punktu synchronizacji. Aby operacja put była wykonywana w ramach elementu sterującego syncpoint, należy użyć wartości MQPMO_SYNCPOINT w polu *Options* struktury MQPMO podczas wywoływania komendy MQPUT. W przypadku operacji pobierania należy użyć wartości MQGMO_SYNCPOINT w polu *Options* struktury MQGMO. Jeśli użytkownik nie wybierze jawnie opcji, działanie domyślne zależy od platformy:

- Multi Wartością domyślną elementu sterującego syncpoint jest NO.
- z/OS Wartością domyślną elementu sterującego syncpoint jest YES.

Gdy wywołanie MQPUT1 jest wysyłane z MQPMO_SYNCPOINT, domyślne zachowanie zmienia się tak, że operacja put jest wykonywana asynchronicznie. Może to spowodować zmianę w zachowaniu niektórych aplikacji, które polegają na zwróconych określonych polach w strukturach MQOD i MQMD, ale które teraz zawierają niezdefiniowane wartości. Aplikacja może określić MQPMO_SYNC_RESPONSE, aby upewnić się, że operacja put jest wykonywana synchronicznie i że wszystkie odpowiednie wartości pól są zakończone.

Jeśli aplikacja otrzymuje kod przyczyny MQRC_BACKED_OUT w odpowiedzi na operację MQPUT lub MQGET w punkcie synchronizacji, aplikacja powinna normalnie wycofać bieżącą transakcję za pomocą MQBACK, a następnie, w razie potrzeby, ponowić próbę wykonania całej transakcji. Jeśli aplikacja odbierze wywołanie MQRC_BACKED_OUT w odpowiedzi na wywołanie MQCMIT lub MQDISC, nie ma potrzeby wywoływania komendy MQBACK.

Za każdym razem, gdy tworzona jest wycofana kopia zapasowa wywołania MQGET, pole *BackoutCount* struktury MQMD komunikatu, którego dotyczy to komunikat, jest zwiększane. Wysoka wartość *BackoutCount* wskazuje komunikat, który został wielokrotnie wycofany. Może to wskazywać na problem z tym komunikatem, który powinien zostać zbadany. Więcej informacji na temat parametru *BackoutCount*zawiera sekcja BackoutCount .

Z wyjątkiem zadania wsadowego z/OS z RRS, jeśli program zgłasza wywołanie MQDISC w czasie, gdy istnieją niezatwierdzone żądania, wystąpi niejawny punkt synchronizacji. Jeśli program zakończy działanie w sposób nieprawidłowy, wystąpi niejawne wyjście.

z/OS W przypadku systemu z/OSniejawny punkt synchronizacji występuje również wtedy, gdy program kończy się normalnie bez wywoływania pierwszego wywołania MQDISC. Program uznaje się za zakończony normalnie, jeśli TCB połączony z MQ zakończy się normalnie. W przypadku uruchamiania w systemie z/OS UNIX System Services and Language Environment (LE) domyślna obsługa warunku jest wywoływana w przypadku abends lub sygnałów. Procedury obsługi warunku LE przetwarzają warunek błędu, a TCB kończy się normalnie. W tych warunkach program MQ zatwierdza jednostkę pracy. Więcej informacji na ten temat zawiera sekcja Wprowadzenie do obsługi warunku środowiska językowego.

z/OS W przypadku programów IBM MQ for z/OS można użyć opcji MQGMO_MARK_SKIP_BACKOUT w celu określenia, że komunikat nie może być wycofany, jeśli wystąpi wycofanie (w celu uniknięcia pętli *MQGET-error-backout*). Więcej informacji na temat korzystania z tej opcji zawiera sekcja “Pomijanie wycofania” na stronie 891.

Zmiany atrybutów kolejki (przez wywołanie MQSET lub przez komendy) nie są naruszane przez zatwierdzanie lub wycofywanie jednostek pracy.

z/OS *Punkty synchronizacji w aplikacjach produktu IBM MQ for z/OS*

W tym temacie opisano sposób korzystania z punktów synchronizacji w menedżerze transakcji (CICS i IMS) i aplikacji wsadowych.

z/OS *Punkty synchronizacji w produkcie CICS Transaction Server for z/OS*

W aplikacji CICS należy utworzyć punkt synchronizacji przy użyciu 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 podręcznik *CICS Application Programming Reference*.

Jeśli w jednostce pracy zaangażowane są inne zasoby odtwarzalne, menedżer kolejek (w połączeniu z menedżerem synchronizacji punktu CICS) uczestniczy w protokole zatwierdzania dwufazowego. W przeciwnym razie menedżer kolejek wykonuje proces zatwierdzania jednofazowego.

Jeśli aplikacja CICS zgłasza wywołanie MQDISC, niejawny punkt synchronizacji nie jest podejmowany. Jeśli aplikacja zostanie zamknięta normalnie, wszystkie otwarte kolejki zostaną zamknięte, a niejawne zatwierdzenie zostanie wykonane. Jeśli aplikacja zostanie zamknięta nieprawidłowo, wszystkie otwarte kolejki są zamykane i nastąpi niejawne wycofanie.

z/OS *Punkty synchronizacji w aplikacjach produktu 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 wprowadzone od poprzedniego punktu kontrolnego, można użyć wywołania komendy IMS ROLB (wycofanie zmian). Więcej informacji na ten temat zawiera dokumentacja produktu IMS .

Menedżer kolejek (w połączeniu z programem IMS syncpoint manager) uczestniczy w protokole zatwierdzania dwufazowego, jeśli w jednostce pracy zaangażowane są również inne zasoby odtwarzalne.

Wszystkie otwarte uchwyty są zamykane przez adapter IMS w punkcie synchronizacji (z wyjątkiem środowiska BMP sterowanego przez zadanie wsadowe lub bez komunikatu). Jest to spowodowane tym, że inny użytkownik może zainicjować następną jednostkę pracy, a sprawdzanie zabezpieczeń produktu IBM MQ jest wykonywane, gdy wykonywane są wywołania MQCONN, MQCONNX i MQOPEN, a nie podczas wykonywania wywołań MQPUT i MQGET.

Jednak w środowisku WFI (Wait-for-Input-WFI) lub pseudo-Wait-for-Input (PWFI) IMS nie jest powiadamiane IBM MQ o zamknięciu uchwytów do momentu otrzymania następnego komunikatu lub do zwrócenia kodu statusu QC do aplikacji. Jeśli aplikacja oczekuje w regionie IMS , a dowolny z tych uchwytów należy do wyzwalanych kolejek, wyzwolenie nie nastąpi, ponieważ kolejki są otwarte. Z tego powodu aplikacje działające w środowisku WFI lub PWFI powinny jawnie MQCLOSE uchwyty kolejki przed przeprowadzką GU do IOPCB dla następnego komunikatu.

Jeśli aplikacja IMS (BMP lub MPP) wysyła wywołanie MQDISC, otwarte kolejki są zamykane, ale nie jest brany żaden niejawny punkt synchronizacji. Jeśli aplikacja zostanie zamknięta normalnie, wszystkie otwarte kolejki zostaną zamknięte, a niejawne zatwierdzenie zostanie wykonane. Jeśli aplikacja zostanie zamknięta nieprawidłowo, wszystkie otwarte kolejki są zamykane i nastąpi niejawne wycofanie.

z/OS *Punkty synchronizacji w aplikacjach wsadowych produktu z/OS*

W przypadku aplikacji wsadowych można używać wywołań zarządzania punktem synchronizacji produktu 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 odtwarzalne usługi menedżera zasobów” na stronie 951.

Zatwierdzanie zmian za pomocą wywołania MQCMIT

Jako dane wejściowe należy podać uchwyt połączenia (*Hconn*), który jest zwracany przez wywołanie MQCONN lub MQCONNX.

Dane wyjściowe komendy MQCMIT są kodem zakończenia i kodem przyczyny. Wywołanie kończy się z ostrzeżeniem, jeśli punkt synchronizacji został zakończony, ale menedżer kolejek wycofał operacje put i get od poprzedniego punktu synchronizacji.

Pomyślne zakończenie wywołania MQCMIT wskazuje menedżerowi kolejek, że aplikacja osiągnęła punkt synchronizacji i że wszystkie operacje put i get wykonywane od poprzedniego punktu synchronizacji zostały wykonane na stałe.

Nie wszystkie odpowiedzi niepowodzenia oznaczają, że program MQCMIT nie został zakończony. Na przykład aplikacja może odbierać MQRC_CONNECTION_BROKEN.

W produkcie MQCMITznajduje się opis wywołania MQCMIT.

Wycofuje zmiany przy użyciu wywołania MQBACK

Jako dane wejściowe należy podać uchwyt połączenia (*Hconn*). Użyj uchwytu, który jest zwracany przez wywołanie MQCONN lub MQCONNX.

Wyjście MQBACK jest kodem zakończenia i kodem przyczyny.

Dane wyjściowe wskazują menedżerowi kolejek, że aplikacja osiągnęła punkt synchronizacji, oraz że wszystkie operacje pobierania i umieszczania zostały wykonane od momentu utworzenia kopii zapasowej ostatniego punktu synchronizacji.

W produkcie MQBACKznajduje się opis wywołania MQBACK.

Usługi zarządzania transakcjami i odtwarzalne usługi menedżera zasobów

Usługi zarządzania transakcjami i odtwarzalnymi usługami menedżera zasobów (RRS) to narzędzie z/OS służące do obsługi dwufazowego punktu synchronizacji w ramach uczestniczących menedżerów zasobów.

Aplikacja może aktualizować zasoby odtwarzalne zarządzane przez różne menedżery zasobów z/OS , takie jak IBM MQ i Db2, a następnie zatwierdzać lub wycofać te aktualizacje jako pojedynczą jednostkę pracy. RRS zapewnia 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 w produkcie IBM MQ for z/OS RRS umożliwia aplikacjom IBM MQ w środowiskach wsadowych, TSO i Db2 aktualizowanie zasobów zarówno IBM MQ , jak i innych niżIBM MQ (na przykład Db2) w ramach jednej logicznej jednostki pracy. Informacje na temat obsługi uczestników usługi RRS można znaleźć w sekcji *Programowanie w systemie z/OS MVS : odtwarzanie zasobów*.

Aplikacja IBM MQ może używać połączeń MQCMIT i MQBACK albo równoważnych wywołań RRS, SRRCMIT i SRRBACK. Więcej informacji zawiera temat “Adapter zadania wsadowego RRS” na stronie 986.

Dostępność RRS

Jeśli produkt RRS nie jest aktywny w systemie z/OS , wszystkie wywołania produktu IBM MQ wysłane z programu powiązanego z kodem pośredniczonym RRS (CSQBRSTB lub CSQBRRSI) zwracają wartość MQRC_ENVIRONMENT_ERROR.

Db2Procedury składowane

Jeśli używane są procedury składowane Db2 z usługą RRS, należy pamiętać o następujących przypadkach:

- Procedury składowane Db2 , które korzystają z usługi RRS, muszą być zarządzane przez menedżer obciążenia (zarządzany przez produkt WLM-zarządzany).

- Jeśli Db2-zarządzana procedura składowana zawiera wywołania produktu IBM MQ i jest ona powiązana z kodem pośredniczonym RRS (CSQBRSTB lub CSQBRRSI), wywołanie MQCONN lub MQCONNX zwraca wartość MQRC_ENVIRONMENT_ERROR.
- Jeśli procedura składowana zarządzana przez WLM zawiera wywołania programu IBM MQ i jest powiązana z kodem pośredniczonym innym niż RRS, wywołanie MQCONN lub MQCONNX zwraca wartość MQRC_ENVIRONMENT_ERROR, chyba że jest to pierwsze wywołanie produktu IBM MQ wykonane od momentu uruchomienia przestrzeni adresowej procedury składowanej.
- Jeśli procedura składowana produktu Db2 zawiera wywołania produktu IBM MQ i jest powiązana z kodem pośredniczonym innym niż RRS, zasoby produktu IBM MQ aktualizowane w tej procedurze składowanej nie są zatwierdzane do momentu zakończenia przestrzeni adresowej procedury składowanej lub do momentu wykonania kolejnej procedury składowanej MQCMIT (przy użyciu kodu pośredniczącego IBM MQ Batch/TSO).
- Wiele kopii tej samej procedury składowanej może być wykonywanych współbieżnie w tej samej przestrzeni adresowej. Ensure that your program is coded in a reentrant manner if you want Db2 to use a single copy of your stored procedure. W przeciwnym razie może zostać wyświetlony komunikat MQRC_HCONN_ERROR w dowolnym wywołaniu programu IBM MQ w programie.
- Nie należy kodować MQCMIT ani MQBACK w procedurze składowanej Db2 zarządzanej przez WLM.
- Zaprojektuj wszystkie programy do uruchomienia w środowisku językowym (LE).

IBM i *Punkty synchronizacji w produkcie CICS dla aplikacji IBM i*

IBM MQ for IBM i uczestniczy w CICS dla IBM i jednostek pracy. Interfejsu MQI można używać w aplikacji CICS dla aplikacji IBM i w celu umieszczania i pobierania komunikatów w bieżącej jednostce pracy.

Za pomocą komendy EXEC CICS SYNCPOINT można ustanowić punkt synchronizacji, który zawiera operacje IBM MQ for IBM i . Aby wycofać wszystkie zmiany w górę do poprzedniego punktu synchronizacji, można użyć komendy EXEC CICS SYNCPOINT ROLLBACK.

If you use MQPUT, MQPUT1, or MQGET with the MQPMO_SYNCPOINT, or MQGMO_SYNCPOINT, option set in a CICS for IBM i application, you cannot log off CICS for IBM i until IBM MQ for IBM i has removed its registration as an API commitment resource. Zatwierdź lub odzyskaj oczekujące operacje umieszczania lub pobierania przed rozłączeniem się z menedżerem kolejek. Pozwala to wylogować się z programu CICS dla produktu IBM i.

Multi *Punkty synchronizacji w produkcie IBM MQ for Multiplatforms*

Obsługa punktu synchronizacji jest wykonywana na dwóch typach jednostek pracy: lokalnym i globalnym.

lokalna jednostka pracy to jedna, w której jedynymi aktualizowanymi zasobami są te, które są aktualizowane przez menedżera kolejek produktu IBM MQ . W tym przypadku koordynacja punktu synchronizacji jest udostępniana przez sam menedżer kolejek przy użyciu procedury zatwierdzania jednofazowego.

Globalne jednostki pracy to jedna z nich, w której aktualizowane są również zasoby należące do innych menedżerów zasobów, takich jak bazy danych. Produkt IBM MQ może koordynować takie jednostki pracy. Mogą być one również koordynowane przez zewnętrzny kontroler zobowiązań. Na przykład:

- Inny menedżer transakcji
- IBM i Kontroler transakcji IBM i

W celu zapewnienia pełnej integralności należy użyć procedury zatwierdzania dwufazowego. Zatwierdzanie dwufazowe może być udostępniane przez menedżery transakcji i bazy danych zgodne z interfejsem XA. Na przykład:

- TXSeries
- System UDB
- IBM i kontroler transakcji IBM i

ULW Produkty IBM MQ mogą koordynować globalne jednostki pracy, korzystając z dwufazowego procesu zatwierdzania.

IBM i Produkt IBM MQ for IBM i może pełnić rolę menedżera zasobów dla globalnych jednostek pracy w środowisku produktu WebSphere Application Server , ale nie może działać jako menedżer transakcji.

Niejawny punkt synchronizacji

V 9.1.0

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 jest lepiej wykonywanych, jeśli te aplikacje używają punktu synchronizacji. Jest to spowodowane tym, że w kolejce jest mniej rywalizacji, jeśli do umieszczania trwałych komunikatów używany jest punkt synchronizacji.

Produkt **ImplSyncOpenOutput** dodaje niejawny punkt synchronizacji, gdy aplikacje umieszczają komunikaty trwałe poza punktem synchronizacji. Zapewnia to poprawę wydajności, bez wiedzy o niejawnym punkcie synchronizacji.

Niejawny punkt synchronizacji zapewnia tylko zwiększenie wydajności, gdy istnieje wiele aplikacji przeznaczonych do umieszczenia w kolejce, ponieważ zmniejsza rywalizację o kolejkę. Oznacza to, że **ImplSyncOpenOutput** określa minimalną liczbę aplikacji, które mają otwartą kolejkę dla danych wyjściowych przed dodanym niejawnym punktem synchronizacji. Domyślna wartość to 2. Oznacza to, że jeśli nie zostanie określony parametr **ImplSyncOpenOutput**, niejawny punkt synchronizacji zostanie dodany tylko wtedy, gdy do kolejki zostanie wstawionych wiele aplikacji.

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. Koordynowanie punktu synchronizacji jest udostępniane przez sam menedżer kolejek (koordynacja wewnętrzna) przy użyciu procesu zatwierdzania jednofazowego.

Aby uruchomić lokalną jednostkę pracy, aplikacja generuje żądania MQGET, MQPUT lub MQPUT1 , określając odpowiednią opcję punktu synchronizacji. Jednostka pracy jest zatwierdzana za pomocą komendy MQCMIT lub wycofana przy użyciu komendy MQBACK. Jednak jednostka pracy kończy się również wtedy, gdy połączenie między aplikacją a menedżerem kolejek jest zerwane, umyślnie lub niezamierzone.

Jeśli aplikacja rozłącza się (MQDISC) z menedżera kolejek, podczas gdy globalna jednostka pracy koordynowana przez produkt IBM MQ jest nadal aktywna, podejmowana jest próba zatwierdzenia jednostki pracy. Jeśli jednak aplikacja kończy działanie bez rozłączania, jednostka pracy zostanie wycofana, ponieważ aplikacja zostanie uznana za zakończonej nieprawidłowo.

Globalne jednostki pracy na wielu platformach
Globalne jednostki pracy należy używać wtedy, gdy konieczne jest uwzględnienie aktualizacji zasobów należących do innych menedżerów zasobów.

W tym przypadku koordynacja może być wewnętrzna lub zewnętrzna względem menedżera kolejek:

Wewnętrzna koordynacja punktu synchronizacji

Koordynacja menedżera kolejek globalnych jednostek pracy nie jest obsługiwana przez produkt IBM MQ for IBM i ani IBM MQ for z/OS. Nie jest on obsługiwany w środowisku IBM MQ MQI client.

W tym miejscu IBM MQ wykonuje koordynację. Aby uruchomić globalną jednostkę pracy, aplikacja wysyła wywołanie MQBEGIN.

Jako dane wejściowe dla 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 generuje żądania MQGET, MQPUT lub MQPUT1 , określając odpowiednią opcję punktu synchronizacji. Oznacza to, że można użyć komendy MQBEGIN w celu zainicjowania globalnej jednostki pracy, która aktualizuje zasoby lokalne, zasoby należące do innych menedżerów zasobów lub obie te wartości. Aktualizacje zasobów należących do innych menedżerów zasobów są dokonywane przy użyciu interfejsu API tego menedżera zasobów. Nie można jednak używać interfejsu MQI do aktualizowania kolejek należących do innych menedżerów kolejek. Wydaj komendę MQCMIT lub MQBACK przed uruchomieniem kolejnych jednostek pracy (lokalne lub globalne).

Globalna jednostka pracy jest zatwierdzana za pomocą komendy MQCMIT; inicjuje to dwufazowe zatwierdzanie wszystkich menedżerów zasobów zaangażowanych w jednostkę pracy. Proces zatwierdzania dwufazowego jest używany, w którym menedżerowie 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 są proszeni o zatwierdzenie. Jeśli dowolny menedżer zasobów sygnalizuje, że nie może zatwierdzić, to każdy z nich jest proszony o wyjście. Alternatywnie można użyć komendy MQBACK, aby wycofać zmiany wszystkich menedżerów zasobów.

Jeśli aplikacja rozłącza się (MQDISC), podczas gdy globalna jednostka pracy jest nadal aktywna, jednostka pracy jest zatwierdzana. Jeśli jednak aplikacja kończy działanie bez rozłączania, jednostka pracy zostanie wycofana, ponieważ aplikacja zostanie uznana za zakończonej nieprawidłowo.

Dane wyjściowe komendy MQBEGIN to kod zakończenia i kod przyczyny.

Jeśli do uruchomienia globalnej jednostki pracy używana jest opcja MQBEGIN, uwzględniane są wszystkie zewnętrzne menedżery zasobów, które zostały skonfigurowane z menedżerem kolejek. Wywołanie uruchamia jednak jednostkę pracy, ale kończy się ostrzeżeniem, jeśli:

- Brak uczestniczących menedżerów zasobów (oznacza to, że żaden menedżer zasobów nie został skonfigurowany z menedżerem kolejek)

lub wersji

- Jeden lub większa liczba menedżerów zasobów nie jest dostępna.

W takich przypadkach jednostka pracy musi zawierać aktualizacje tylko tych menedżerów zasobów, które były dostępne w momencie uruchomienia jednostki pracy.

Jeśli jeden z menedżerów zasobów nie może zatwierdzić swoich aktualizacji, wszystkie menedżery zasobów są instruowane, aby wycofać ich aktualizacje, a program MQCMIT kończy działanie z ostrzeżeniem. W nietypowych okolicznościach (zwykle interwencja operatora) wywołanie MQCMIT może się nie powieść, jeśli niektórzy menedżerowie zasobów zatwierdzają swoje aktualizacje, ale inne je wycofają; praca jest uważana za zakończonej wynikiem *mieszanym* . Takie wystąpienia są diagnozowane w dzienniku błędów menedżera kolejek, dzięki czemu możliwe jest podjęcie działań zaradczych.

Operacja MQCMIT globalnej jednostki pracy powiedzie się, jeśli wszystkie osoby zarządzające zasobami zaangażują się w ich aktualizacje.

Opis wywołania MQBEGIN można znaleźć w sekcji MQBEGIN.

Zewnętrzna koordynacja punktu synchronizacji

Taka sytuacja ma miejsce wtedy, gdy wybrano koordynatora punktu synchronizacji, innego niż IBM MQ , na przykład CICS, Encina lub Tuxedo.

W takiej sytuacji produkt IBM MQ w systemach UNIX and Linux i IBM MQ for Windows zarejestrują swoje zainteresowania w wyniku jednostki pracy z koordynatorem punktu synchronizacji, dzięki czemu mogą one zatwierdzić lub wycofać wszystkie niezatwierdzone operacje get lub put w zależności od potrzeb. Zewnętrzny koordynator punktu synchronizacji określa, czy są dostępne jedno-lub dwufazowe protokoły zobowiązań.

W przypadku korzystania z zewnętrznego koordynatora, MQCMIT, MQBACK i MQBEGIN nie mogą zostać wydane. Wywołania tych funkcji nie powiodą się z powodu kodu 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 wywołanie jawne, a zostanie wydana wywołanie MQPUT z opcją MQPMO_SYNCPOINT, gdy jednostka pracy nie jest uruchomiona, zwracany jest kod zakończenia MQRC_SYNCPOINT_NOT_AVAILABLE.

Zakres jednostki pracy jest określany przez koordynatora punktu synchronizacji. Stan połączenia między aplikacją a menedżerem kolejek wpływa na powodzenie lub niepowodzenie wywołań MQI, które dotyczą aplikacji, a nie stanu jednostki pracy. Aplikacja może na przykład rozłączyć się i ponownie połączyć się z menedżerem kolejek podczas aktywnej jednostki pracy i wykonać dalsze operacje MQGET i MQPUT w tej samej jednostce pracy. Jest to znane jako oczekiwanie na rozłączenie.

Można używać wywołań funkcji API produktu IBM MQ w programach CICS , niezależnie od tego, czy mają być używane możliwości interfejsu XA w produkcie CICS. Jeśli nie jest używany interfejs XA, operacje umieszczania i pobierania komunikatów z kolejkami i z kolejek nie będą zarządzane w ramach CICS jednostek niepodzielnych. Jedną z przyczyn wyboru tej metody jest to, że ogólna spójność jednostki pracy nie jest dla Ciebie istotna.

Jeśli integralność jednostek pracy jest dla Ciebie ważna, należy użyć interfejsu XA. Jeśli używany jest interfejs XA, produkt CICS korzysta z protokołu zatwierdzania dwufazowego w celu zapewnienia, że wszystkie zasoby w obrębie jednostki pracy są aktualizowane razem.

Więcej informacji na temat konfigurowania obsługi transakcyjnej zawiera sekcja Scenariusze obsługi transakcyjnej, a także dokumentacja produktu TXSeries CICS , na przykład *Podręcznik administrowania produktem TXSeries for Multiplatforms CICS dla systemów otwartych*.

Multi V 9.1.0 *Niejawny punkt synchronizacji na wielu platformach*

Niejawna obsługa punktu synchronizacji umożliwia umieszczanie komunikatów trwałych poza punktem synchronizacji.

Podczas umieszczania trwałych komunikatów produkt IBM MQ jest zoptymalizowany pod kątem umieszczania trwałych komunikatów w punkcie synchronizacji. Wielokrotne aplikacje współbieżnie umieszczające komunikaty trwałe w tej samej kolejce zwykle są wykonywane lepiej, jeśli te aplikacje używają punktu synchronizacji. Jest to spowodowane tym, że strategia blokowania produktu IBM MQ jest bardziej wydajna, jeśli podczas umieszczania trwałych komunikatów używany jest punkt synchronizacji.

Parametr **ImplSyncOpenOutput** w pliku `qm.ini` określa, czy można dodać niejawny punkt synchronizacji w przypadku, gdy aplikacje umieszczają komunikaty trwałe poza punktem synchronizacji. Może to zapewnić poprawę wydajności, bez wiedzy o niejawnym punkcie synchronizacji.

Niejawny punkt synchronizacji zapewnia tylko zwiększenie wydajności, gdy istnieje wiele aplikacji jednocześnie wprowadzających do kolejki, ponieważ zmniejsza rywalizację o blokadę. **ImplSyncOpenOutput** określa minimalną liczbę aplikacji, które mają otwartą kolejkę dla danych wyjściowych, zanim można dodać niejawny punkt synchronizacji. Wartością domyślną jest *2*. Oznacza to, że jeśli użytkownik nie określi jawnie opcji **ImplSyncOpenOutput**, niejawny punkt synchronizacji zostanie dodany tylko wtedy, gdy do kolejki zostanie wstawionych wiele aplikacji.

Jeśli zostanie dodany niejawny punkt synchronizacji, statystyki będą odzwierciedlały to zdarzenie, a dane wyjściowe transakcji mogą zostać wyświetlone w programie **runmqsc display conn**.

Ustaw **ImplSyncOpenOutput**=*WYŁĄCZONE* , jeśli nigdy nie chcesz dodać niejawnego punktu synchronizacji.

Więcej informacji na ten temat zawiera sekcja Parametry strojenia .

Interfejsy do zewnętrznych menedżerów punktów synchronizacji w wielu platformach

Produkt IBM MQ for Multiplatforms obsługuje koordynację transakcji przez zewnętrzne menedżery synchronizacji, które korzystają z interfejsu XA X/Open.

Niektóre menedżery transakcji XA (TXSeries) wymagają, aby każdy menedżer zasobów XA udostępniał swoją nazwę. Jest to łańcuch o nazwie name w strukturze przełącznika XA.

- ULW Menedżer zasobów dla IBM MQ w systemie UNIX, Linux, and Windows ma nazwę MQSeries_XA_RMI.
- IBM i W przypadku bazy danych IBM inazwą menedżera zasobów jest MQSeries XA RMI.

Więcej szczegółów na temat interfejsów XA można znaleźć w dokumentacji XA *CAE Specification Distributed Transaction Processing: The XA Specification*(Specyfikacja transakcji rozproszonych: Specyfikacja XA) opublikowanej przez grupę Open Group.

W konfiguracji interfejsu 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 wycofanie transakcji w obu menedżerach zasobów. W ten sposób działa on dla statycznie zarejestrowanego menedżera zasobów:

1. Aplikacja powiadamia koordynatora punktu synchronizacji o tym, że chce uruchomić transakcję.
2. Koordynator punktu synchronizacji wysyła wywołanie do wszystkich menedżerów zasobów, o których wie, w celu powiadomienia ich o bieżącej transakcji.
3. Problemy aplikacji wywołują aktualizację zasobów zarządzanych przez menedżery zasobów powiązane z bieżącą transakcją.
4. Aplikacja żąda, aby koordynator punktu synchronizacji zatwierdził transakcję lub wycofał transakcję.
5. Koordynator punktu synchronizacji wywołuje każdy menedżer zasobów przy użyciu protokołów zatwierdzania dwufazowego w celu zakończenia transakcji zgodnie z żądaniem.

Specyfikacja XA wymaga, aby każdy menedżer zasobów udostępnił strukturę o nazwie Przełącznik XA. Ta struktura deklaruje możliwości menedżera zasobów oraz funkcje, które mają być wywoływane przez koordynatora punktu synchronizacji.

Istnieją dwie wersje tej struktury:

Tabela 132. Wersje przełącznika XA

Wersja	**Opis**
MQRMIXASwitch	Zarządzanie statycznymi zasobami XA
MQRMIXASwitchDynamic	Dynamiczne zarządzanie zasobami XA

Aby uzyskać listę bibliotek zawierających tę strukturę, należy zapoznać się z strukturą przełącznika XA IBM MQ.

Metoda, która musi być używana do łączenia ich z koordynatorem punktu synchronizacji XA, jest zdefiniowana przez koordynatora. Aby określić, w jaki sposób produkt IBM MQ ma współpracować z koordynatorem punktu synchronizacji XA, należy zapoznać się z dokumentacją udostępnionym przez tego koordynatora.

Struktura *xa_info* , która jest przekazywana do dowolnego wywołania *xa_open* przez koordynatora punktu synchronizacji, może być nazwą menedżera kolejek, który ma być administrowany. Ten sam formularz ma taką samą postać, jak nazwa menedżera kolejek przekazanego do tabeli MQCONN lub MQCONNX. Jeśli domyślny menedżer kolejek ma być używany, może on być pusty. Można jednak użyć dwóch dodatkowych parametrów TPM i AXLIB.

Opcja TPM umożliwia określenie IBM MQ nazwy menedżera transakcji, na przykład CICS. AXLIB pozwala na określenie rzeczywistej nazwy biblioteki w menedżerze transakcji, w którym znajdują się punkty wejścia XA AX.

Jeśli używany jest dowolny z tych parametrów lub inny niż domyślny menedżer kolejek, należy określić nazwę menedżera kolejek przy użyciu parametru QMNAME. Więcej informacji na ten temat zawiera sekcja Parametry CHANNEL, TRPTYPE, CONNAME i QMNAME w łańcuchu xa_open.

Ograniczenia

1. Globalne jednostki pracy nie są dozwolone ze współużytkowanym serwerem Hconn (zgodnie z opisem w sekcji “Połączenia współużytkowane (niezależne od wątku) z produktem MQCONNX” na stronie 830).
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 deklarowane jako funkcje _cdecl.
4. Zewnętrzny koordynator punktu synchronizacji może w danym momencie administrować tylko jednym menedżerem kolejek. Jest to spowodowane tym, że koordynator ma efektywne połączenie z każdym menedżerem kolejek i w związku z tym podlega regułce, że tylko jedno połączenie jest dozwolone w danym momencie.

   Uwaga: Uwaga: Aplikacja kliencka JMS (aplikacja CLIENT JEE) działająca na serwerze JEE nie ma tego ograniczenia, tak więc 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 regułce, z której dozwolony jest tylko jedno połączenie w danym momencie.
5. Wszystkie aplikacje, które są uruchamiane za pomocą koordynatora punktu synchronizacji, mogą łączyć się tylko z menedżerem kolejek, który jest administrowany przez koordynatora, ponieważ są one już efektywnie połączone z tym menedżerem kolejek. Muszą one wydać komendę MQCONN lub MQCONNX w celu uzyskania uchwytu połączenia i musi wydać komendę MQDISC przed wyjściem z systemu. Alternatywnie mogą one używać wyjścia UE014015 dla TXSeries CICS.

IBM i *Interfejsy do zewnętrznego menedżera synchronizacji punktu synchronizacji produktu IBM i*

Produkt IBM MQ for IBM i może używać rodzimej kontroli transakcji produktu 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 w produkcie IBM izawiera publikacja *IBM i Programming: Backup and Recovery Guide, SC21-8079* .

Aby uruchomić narzędzia kontroli transakcji produktu IBM i , należy użyć komendy systemowej STRCMTCTL. Aby zakończyć kontrolę transakcji, należy użyć komendy systemowej ENDCMTCTL.

Uwaga: Wartością domyślną pola *Zasięg definicji transakcji* jest *ACTGRP. Wartość ta musi być zdefiniowana jako *JOB dla IBM MQ dla IBM i. Na przykład:

```
STRCMTCTL LCKLVL(*ALL) CMTSCOPE(*JOB)
```

Produkt IBM MQ for IBM i może również wykonywać lokalne jednostki pracy zawierające tylko aktualizacje zasobów produktu IBM MQ . Wybór między lokalnymi jednostkami pracy a uczestnictwem w globalnych jednostkach pracy koordynowanych przez produkt IBM i jest dokonany w każdej aplikacji, gdy aplikacja wywołuje operację MQPUT, MQPUT1lub MQGET, określając parametr MQPMO_SYNCPOINT lub MQGMO_SYNCPOINT lub MQBEGIN. Jeśli kontrola transakcji nie jest aktywna, gdy pierwsze takie wywołanie zostanie wydane, program IBM MQ uruchomi lokalną jednostkę pracy, a wszystkie dalsze jednostki pracy dla tego połączenia z programem IBM MQ korzystają również z lokalnych jednostek pracy, niezależnie od tego, czy kontrola transakcji została następnie uruchomiona. Aby zatwierdzić lokalną jednostkę pracy, użyj komendy MQCMIT. Aby utworzyć kopię zapasową lokalnej jednostki pracy, należy użyć komendy MQBACK. Wywołania funkcji zatwierdzania i wycofywania zmian w produkcie IBM i , takie jak komenda CL COMMIT, nie mają wpływu na lokalne jednostki pracy systemu IBM MQ .

Jeśli jako zewnętrzny koordynator punktu synchronizacji ma być używany produkt IBM MQ for IBM i z rodzimą kontrolą transakcji produktu IBM i , należy upewnić się, że wszystkie zadania z kontrolą transakcji są aktywne i że w zadaniu jednowątkowym używany jest produkt IBM MQ .

W przypadku wywołania MQPUT, MQPUT1lub MQGET, określając wartość MQPMO_SYNCPOINT lub MQGMO_SYNCPOINT, w zadaniu wielowątkowym, w którym uruchomiono kontrolę transakcji, wywołanie kończy się niepowodzeniem z kodem przyczyny MQRC_SYNCPOINT_NOT_AVAILABLE.

Możliwe jest użycie lokalnych jednostek pracy, a wywołania MQCMIT i MQBACK w zadaniu wielowątkowym.

W przypadku wywołania MQPUT, MQPUT1lub MQGET, określając MQPMO_SYNCPOINT lub MQGMO_SYNCPOINT, po uruchomieniu kontroli transakcji, produkt IBM MQ for IBM i dodaje siebie jako zasób zatwierdzania interfejsu API do definicji kontroli transakcji. Zwykle jest to pierwsze tego typu wywołanie w zadaniu. Chociaż istnieją jakiekolwiek zasoby zatwierdzania transakcji zarejestrowane w ramach określonej definicji kontroli transakcji, nie można zakończyć kontroli transakcji dla tej definicji.

Produkt IBM MQ for IBM i usuwa jego rejestrację jako zasób transakcji API po rozłączeniu z menedżerem kolejek, jeśli w bieżącej jednostce pracy nie ma oczekujących operacji MQI.

W przypadku rozłączenia się 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 transakcji API, dzięki czemu zostanie powiadomiony o następnym zatwierdzeniu lub wycofaniu. Po osiągnięciu następnego punktu synchronizacji program IBM MQ for IBM i zatwierdza lub wycofuje zmiany zgodnie z wymaganiami. Aplikacja może rozłączyć się i ponownie połączyć się z menedżerem kolejek podczas aktywnej jednostki pracy i wykonać dalsze operacje MQGET i MQPUT w tej samej jednostce pracy (to jest oczekiwanie na rozłączenie).

Jeśli zostanie podjęta próba wydania komendy systemowej ENDCMTCTL dla tej definicji kontroli transakcji, zostanie wyświetlony komunikat CPF8355 wskazujący, że oczekujące zmiany były aktywne. Ten komunikat pojawia się również w protokole zadania po zakończeniu zadania. Aby tego uniknąć, należy zatwierdzić lub wycofać zmiany wszystkich oczekujących operacji IBM MQ for IBM i i odłączyć się od menedżera kolejek. Dlatego użycie komend COMMIT lub ROLLBACK przed zakończeniem działania komendy ENDCMTCTL powoduje pomyślne zakończenie kontroli transakcji.

Jeśli jako zewnętrzny koordynator punktu synchronizacji używana jest opcja IBM i kontroli transakcji, nie można wywołać wywołań MQCMIT, MQBACK i MQBEGIN. Wywołania tych funkcji nie powiodą się z powodu kodu przyczyny MQRC_ENVIRONMENT_ERROR.

Aby zatwierdzić lub wycofać zmiany (to jest, aby wycofać) jednostkę pracy, należy użyć jednego z języków programowania, który obsługuje kontrolę transakcji. Na przykład:

- Komendy CL: COMMIT i ROLLBACK
- Funkcje programowania ILE C: _Rcommit i _Rrollback
- ILE RPG: COMMIT i ROLBK
- COBOL/400: COMMIT i ROLLBACK

Jeśli jako zewnętrzny koordynator punktu synchronizacji z produktem IBM MQ for IBM iużywana jest kontrola transakcji IBM i , produkt IBM i wykonuje dwufazowy protokół zatwierdzania, 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 to zdarzyć, na przykład, jeśli zadania wewnętrzne menedżera kolejek są zakończone. W takiej sytuacji dziennik zadania wykonujący zatwierdzenie zawiera komunikat CPF835F wskazujący, że operacja zatwierdzania lub wycofania zmian nie powiodła się. Komunikaty poprzedzające tę informację wskazują przyczynę problemu, niezależnie od tego, czy wystąpiły one podczas operacji zatwierdzania, czy wycofania zmian, a także logicznej jednostki pracy (LUWID) dla uszkodzonej jednostki pracy.

Jeśli problem został spowodowany awarią zasobu zobowiązania API IBM MQ podczas zatwierdzania lub wycofywania zmian przygotowanej jednostki pracy, można użyć komendy WRKMQMTRN w celu zakończenia operacji i przywrócenia integralności transakcji. Komenda wymaga, aby użytkownik znał identyfikator LUWID jednostki pracy w celu zatwierdzenia i utworzenia kopii zapasowej.

Uruchamianie aplikacji produktu IBM MQ przy użyciu wyzwalaczy

Informacje na temat wyzwalaczy i sposobów uruchamiania aplikacji IBM MQ przy użyciu wyzwalaczy.

Niektóre aplikacje produktu IBM MQ , które obsługują kolejki, działają w sposób ciągły, dlatego są zawsze dostępne do pobierania komunikatów, które docierają do kolejek. Być może jednak nie będzie to pożądane, gdy liczba komunikatów przychodzących do kolejek jest nieprzewidywalna. W takim przypadku aplikacje mogą pochłaniać zasoby systemowe nawet wtedy, gdy nie ma żadnych komunikatów do pobrania.

Produkt IBM MQ udostępnia narzędzie, które umożliwia automatyczne uruchomienie aplikacji, gdy dostępne są komunikaty do pobrania. Ten obiekt jest znany jako *wyzwalanie*.

Więcej informacji na temat kanałów wyzwalających zawiera sekcja Wyzwalanie kanałów.

Co to jest wyzwalanie?

Menedżer kolejek definiuje pewne warunki jako stanowiące *zdarzenia wyzwalające*.

Jeśli wyzwalanie jest włączone dla kolejki i wystąpi zdarzenie wyzwalające, menedżer kolejek wysyła *komunikat wyzwalacza* do kolejki o nazwie *kolejka inicjujowa*. Obecność komunikatu wyzwalacza w kolejce inicjuj. wskazuje, że wystąpiło zdarzenie wyzwalające.

Komunikaty wyzwalacza wygenerowane przez menedżer kolejek nie są trwałe. Zmniejsza to rejestrowanie (skutkuje poprawą wydajności) i minimalizuje duplikaty podczas restartu, a więc poprawia czas restartu.

Program, który przetwarza kolejkę inicjujący, jest nazywany *aplikacją monitora wyzwalacza*, a jego funkcją jest odczytywanie komunikatu wyzwalacza i podjęcie odpowiednich działań w oparciu o informacje zawarte w komunikacie wyzwalacza. Zazwyczaj to działanie polega na uruchomieniu innej aplikacji w celu przetworzenia kolejki, która wygenerował komunikat wyzwalacza. Z punktu widzenia menedżera kolejek nie ma nic specjalnego w aplikacji wyzwalacza-monitor; jest to po prostu inna aplikacja, która odczytuje komunikaty z kolejki (kolejka inicjujący).

Jeśli wyzwalanie jest włączone dla kolejki, można utworzyć *obiekt definicji procesu* powiązany z tą kolejką. Ten obiekt zawiera informacje na temat aplikacji, która przetwarza komunikat, który spowodował zdarzenie wyzwalające. Jeśli tworzony jest obiekt definicji procesu, menedżer kolejek wyodrębnia te informacje i umieszcza je w komunikacie wyzwalaczem w celu użycia przez aplikację monitorującego wyzwalacza. Nazwa definicji procesu powiązana z kolejką jest nadawana przez atrybut lokalnej kolejki produktu *ProcessName* . Każda kolejka określa inną definicję procesu lub kilka kolejek może współużytkować definicję procesu.

Jeśli uruchomienie kanału ma być wyzwalane, nie trzeba definiować obiektu definicji procesu. Zamiast niej używana jest definicja kolejki transmisji.

Wyzwalanie jest obsługiwane przez klienty IBM MQ działające na serwerze UNIX, Linux, and Windows. Aplikacja działająca w środowisku klienta jest taka sama, jak działająca w pełnym środowisku IBM MQ , z tą różnicą, że połączono ją 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 w momencie wyzwalania i gdy spełnione są warunki, wymaga, aby komunikaty wyzwalacza były zapisywane.

Definicja procesu
Z kolejką aplikacji może być powiązany *obiekt definicji procesu* zawierający szczegółowe informacje na temat aplikacji, które będą dostawać komunikaty z kolejki aplikacji. (Lista atrybutów znajduje się w sekcji Atrybuty definicji procesów).

Należy pamiętać, że jeśli wyzwalacz ma uruchamiać kanał, nie ma potrzeby definiowania obiektu definicji procesu.

Kolejka transmisji
Jeśli wyzwalacz ma być uruchamiany w celu uruchomienia kanału, potrzebna jest kolejka transmisji.

W przypadku kolejki transmisji na dowolnej platformie innej niż Linux, atrybut *TriggerData* kolejki transmisji może określać nazwę kanału, który ma zostać uruchomiony. Może to zastąpić definicję

procesu wyzwalającą kanały, ale jest ona używana tylko wtedy, gdy definicja procesu nie została utworzona.

zdarzenie wyzwalające
Zdarzenie wyzwalające to zdarzenie, które powoduje wygenerowanie komunikatu wyzwalacza przez menedżer kolejek. Zwykle jest to komunikat, który przylatuje do kolejki aplikacji, ale może on również wystąpić w innych sytuacjach. Na przykład: “Warunki dla zdarzenia wyzwalającego” na stronie 965.

Produkt IBM MQ zawiera szereg opcji umożliwiających sterowanie warunkami, które powodują zdarzenie wyzwalające (patrz “Sterowanie zdarzeniami wyzwalającymi” na stronie 969).

komunikat wyzwalacza
Menedżer kolejek tworzy *komunikat wyzwalacza* , gdy rozpoznaje zdarzenie wyzwalające. Kopiuje on do komunikatu wyzwalacza informacje o aplikacji, która ma zostać uruchomiona. Informacje te pochodzą z kolejki aplikacji oraz z obiektu definicji procesu powiązanego z kolejką aplikacji.

Komunikaty wyzwalacza mają stały format (patrz “Format komunikatów wyzwalacza” na stronie 977).

Kolejka inicjująca
Kolejka inicjuj. jest kolejką lokalną, w której menedżer kolejek umieszcza komunikaty wyzwalacza. Należy pamiętać, że kolejka inicjujący nie może być kolejką aliasową ani kolejką modelową.

Menedżer kolejek może być właścicielem więcej niż jednej kolejki inicjujowej, a każda z nich jest powiązana z jedną lub większą liczbę kolejek aplikacji.

z/OS Kolejka współużytkowana, kolejka lokalna dostępna dla menedżerów kolejek w grupie współużytkowania kolejek, może być kolejką inicjują w systemie IBM MQ for z/OS.

monitor wyzwalacza
Monitor wyzwalacza to działający w sposób ciągły program, który służy do wykonywania jednej lub większej liczby kolejek inicjuj. Gdy komunikat wyzwalacza przybywa do kolejki inicjującej, monitor wyzwalacza wczytuje ten komunikat. Monitor wyzwalacza używa informacji znajdujących się w komunikacie wyzwalacza. Wysyła ona komendę uruchamiające aplikację, która ma pobierać komunikaty przychodzące do kolejki aplikacji, przekazując informacje zawarte w nagłówku komunikatu wyzwalacza, który zawiera nazwę kolejki aplikacji.

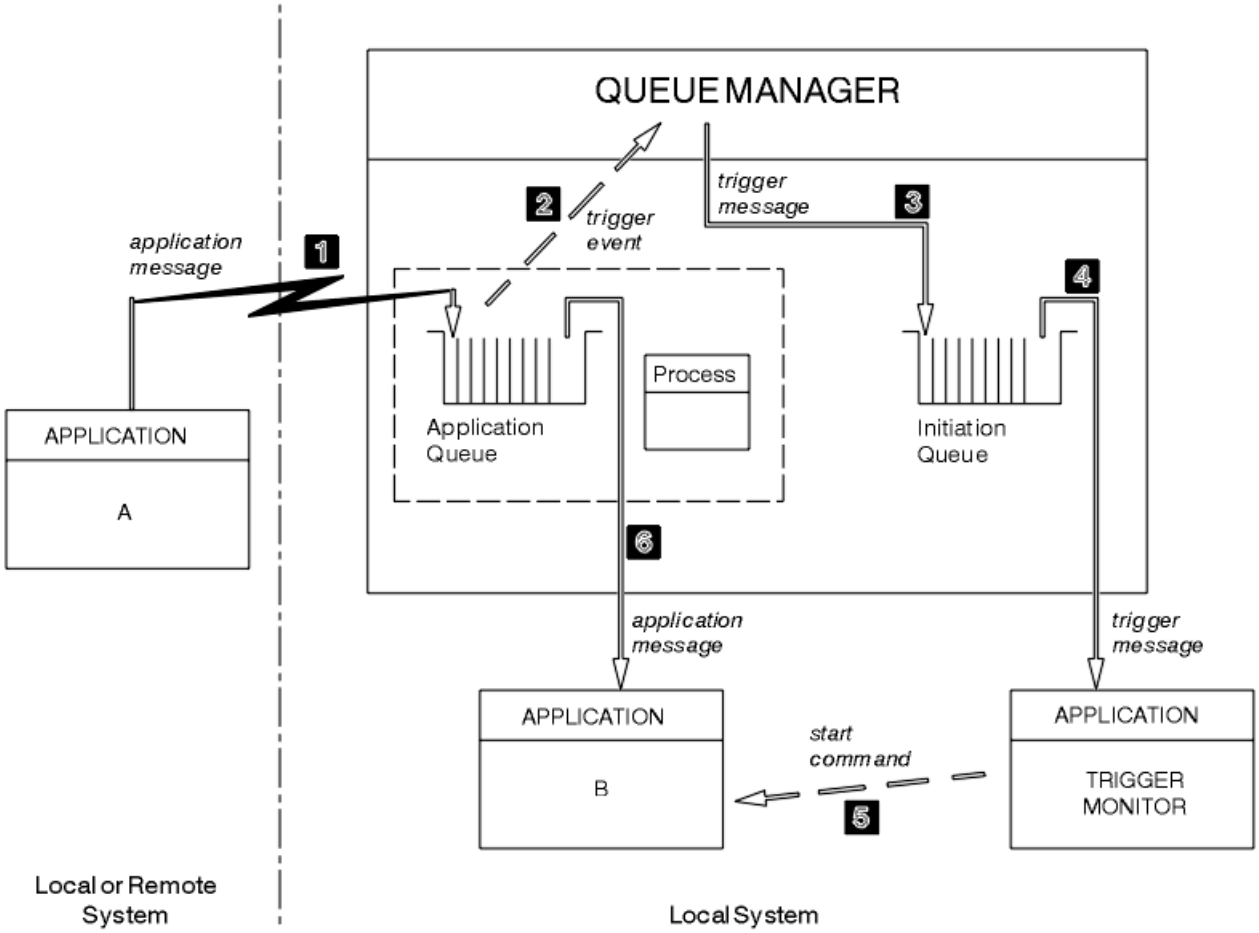
Na wszystkich platformach specjalny monitor wyzwalacza znany jako inicjator kanału jest odpowiedzialny za uruchamianie kanałów.

z/OS W systemie z/OSinicjator kanału jest zwykle uruchamiany ręcznie lub może być uruchamiany automatycznie, gdy menedżer kolejek rozpoczyna się od zmiany CSQINP2 w startowym JCL 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 inicjuj przez monitory wyzwalacza” na stronie 973.

Aby zrozumieć, jak działa wyzwalanie, należy wziąć pod uwagę Rysunek 100 na stronie 961, który jest przykładem typu wyzwalacza FIRST (MQTT_FIRST).



Rysunek 100. Przepływ komunikatów aplikacji i wyzwalaczy

W programie Rysunek 100 na stronie 961sekwencja zdarzeń jest następująca:

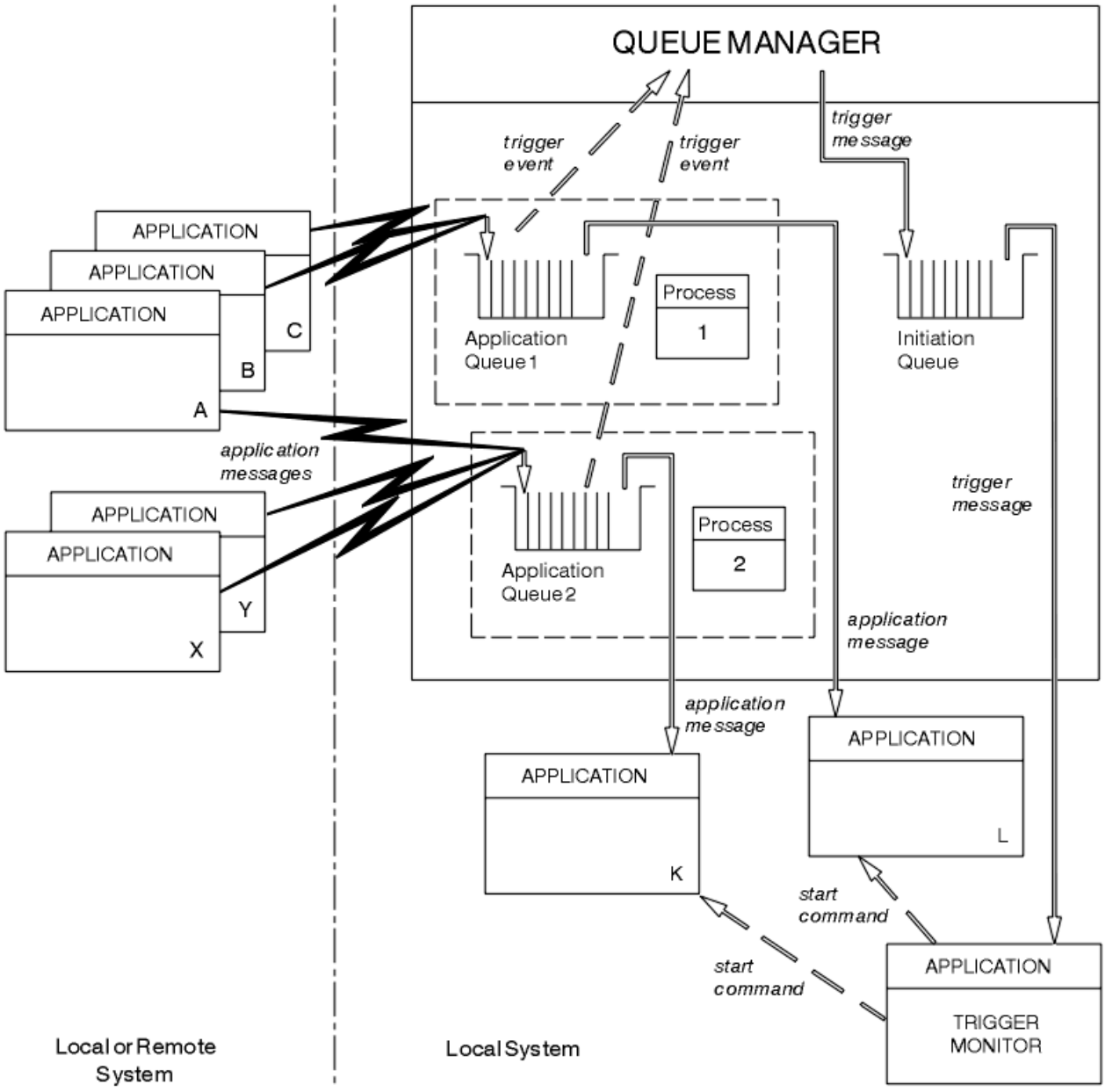
1. Aplikacja A, która może być lokalna lub zdalna w stosunku do menedżera kolejek, umieszcza komunikat w kolejce aplikacji. Żadna aplikacja nie ma tej kolejki otwartej na dane wejściowe. Jednak ten fakt ma znaczenie tylko w przypadku wyzwalaczy typu FIRST i DEPTH.
2. Menedżer kolejek sprawdza, czy spełnione są warunki, na podstawie których musi wygenerować zdarzenie wyzwalające. Są one generowane 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 inicjuj. powiązanej z tą kolejką aplikacji, ale tylko wtedy, gdy aplikacja (monitor wyzwalacza) ma otwartą kolejkę inicjują do wejścia.
4. Monitor wyzwalacza pobiera komunikat wyzwalacza z kolejki inicjuj.
5. Monitor wyzwalacza wysyła komendę do uruchomienia aplikacji B (aplikacji serwera).
6. Aplikacja B otwiera kolejkę aplikacji i pobiera komunikat.

Uwaga:

1. Jeśli kolejka aplikacji jest otwarta dla danych wejściowych, przez dowolny program i ma ustawioną wartość FIRST lub DEPTH, żadne zdarzenie wyzwalające nie będzie miało miejsca, ponieważ kolejka jest już obsługiwana.
2. Jeśli kolejka inicjujący nie jest otwarta dla danych wejściowych, menedżer kolejek nie generuje komunikatów wyzwalacza; oczekuje do momentu otwarcia przez aplikację kolejki inicjujowej dla danych wejściowych.

3. W przypadku wyzwalania dla kanałów należy użyć wyzwalacza typu FIRST lub DEPTH.
4. Wyzwalane aplikacje działają pod identyfikatorem użytkownika i grupą użytkownika, który uruchomił monitor wyzwalacza, użytkownika programu CICS lub użytkownika, który uruchomił menedżer kolejek.

Dotychczas relacja między kolejkami w ramach wyzwalania była tylko na jednej, na jednej podstawie. Rozważ Rysunek 101 na stronie 962.



Rysunek 101. Relacja kolejek w wyzwalaniu

Z kolejką aplikacji jest powiązany obiekt definicji procesu, który zawiera szczegółowe informacje na temat aplikacji, która będzie przetwarzała komunikat. Menedżer kolejek umieszcza informacje w komunikacie wyzwalacza, dlatego konieczna jest tylko jedna kolejka inicjacyjna. Monitor wyzwalacza wyodrębnia te informacje z komunikatu wyzwalacza i uruchamia odpowiednią aplikację w celu zajmowania się komunikatem w każdej kolejce aplikacji.

Należy pamiętać, że aby wyzwolić początek kanału, nie trzeba definiować obiektu definicji procesu. Definicja kolejki transmisji może określić kanał, który ma zostać wyzwolony.

Użyj poniższych odsyłaczy, aby dowiedzieć się więcej na temat uruchamiania aplikacji IBM MQ za pomocą wyzwalaczy:

- “Wymagania wstępne dla wyzwalania” na stronie 963
- “Warunki dla zdarzenia wyzwalającego” na stronie 965
- “Sterowanie zdarzeniami wyzwalającymi” na stronie 969
- “Projektowanie aplikacji, która korzysta z wyzwalanych kolejek” na stronie 972
- “Przetwarzanie kolejki inicjuj przez monitory wyzwalacza” na stronie 973
- “Właściwości komunikatów wyzwalacza” na stronie 976
- “Gdy wyzwalanie nie działa” na stronie 978

Pojęcia pokrewne

“Interfejs kolejki komunikatów-przegląd” na stronie 811
Informacje na temat komponentów interfejsu kolejek komunikatów (Message Queue Interface-MQI).

“Nawiąże połączenie z menedżerem kolejek i odłączenie go od niego” na stronie 824
Aby można było korzystać z usług programistycznych produktu IBM MQ , program musi mieć połączenie z menedżerem kolejek. Informacje zawarte w tej sekcji umożliwiają zapoznanie się z informacjami na temat nawiązywania połączenia z menedżerem kolejek i z jego rozłączeniem.

“Otwieranie i zamykanie obiektów” na stronie 833
Te informacje udostępniają wgląd w otwieranie i zamykanie obiektów produktu IBM MQ .

“Umieszczanie komunikatów w kolejce” na stronie 844
Te informacje umożliwiają zapoznanie się z informacjami na temat umieszczania komunikatów w kolejce.

“Pobieranie komunikatów z kolejki” na stronie 860
Ta sekcja zawiera informacje na temat pobierania komunikatów z kolejki.

“Sprawdzanie i ustawianie atrybutów obiektu” na stronie 943
Atrybuty to właściwości, które definiują parametry obiektu IBM MQ .

“Zatwierdzanie i wycofywanie jednostek pracy” na stronie 946
W tej sekcji opisano sposób zatwierdzania i tworzenia kopii zapasowych wszystkich możliwych do odtworzenia operacji get i put, które wystąpiły w jednostce pracy.

“Praca z interfejsem MQI i klastrami” na stronie 978
Istnieją specjalne opcje dotyczące wywołań i kodów powrotu, które odnoszą się do technologii klastrowej.

“Używanie i zapisywanie aplikacji w systemie IBM MQ for z/OS” na stronie 983
Aplikacje produktu IBM MQ for z/OS mogą być uruchamiane z programów działających w wielu różnych środowiskach. Oznacza to, że mogą skorzystać z udogodnień dostępnych w więcej niż jednym środowisku.

“Aplikacje pomostowe IMS i IMS w systemie IBM MQ for z/OS” na stronie 70
Te informacje ułatwiają pisanie aplikacji produktu IMS przy użyciu produktu IBM MQ.

Wymagania wstępne dla wyzwalania

Te informacje umożliwiają zapoznanie się z krokami, które należy wykonać przed uruchomieniem wyzwalania.

Zanim aplikacja będzie mogła skorzystać z wyzwolenia, wykonaj następujące kroki:

1. Albo:
   a. Utwórz kolejkę inicjujową 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 nazwa SYSTEM.DEFAULT.INITIATION.QUEUE lub, jeśli uruchamiasz kanały z wyzwalaczami, SYSTEM.CHANNEL.INITQ) i podaj jego nazwę w polu *InitiationQName* w kolejce aplikacji.

2. Powiąż kolejkę inicjujową z kolejką aplikacji. Menedżer kolejek może być właścicielem więcej niż jednej kolejki inicjuj. Niektóre kolejki aplikacji mogą być obsługiwane przez różne programy. W takim przypadku można użyć jednej kolejki inicjuj dla każdego programu obsługującego, mimo że 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 znajduje się wyciąg z programu CL dla 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, utwórz obiekt definicji procesu, który będzie zawierał informacje dotyczące aplikacji, która ma obsługiwać kolejkę aplikacji. Na przykład, aby wyzwolić-uruchom transakcję CICS payroll o nazwie PAYR:

```
DEFINE PROCESS (process.name) +
       REPLACE +
       DESCR ('process description') +
       APPLICID ('PAYR') +
       APPLTYPE (CICS) +
       USERDATA ('Payroll data')
```

IBM i Poniżej znajduje się ekstrakt z programu CL dla 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 on informacje z atrybutów obiektu definicji procesu do komunikatu wyzwalacza.

Platforma	Aby utworzyć obiekt definicji procesu
UNIX, Linux, and Windows systemy	Użyj opcji DEFINE PROCESS lub SYSTEM.DEFAULT.PROCESS i zmodyfikuj za pomocą instrukcji ALTER PROCESS
z/OS z/OS	Użyj opcji DEFINE PROCESS (ZDEFINIUJ PROCES) (patrz kod przykładowy w kroku “3” na stronie 964) lub skorzystaj z paneli operacji i sterowania.

Platforma	Aby utworzyć obiekt definicji procesu
IBM i IBM i IBM i	Użyj programu CL zawierającego kod jak w kroku “3” na stronie 964.

4. Opcjonalnie: utwórz definicję kolejki transmisji i użyj odstępów dla atrybutu **ProcessName** .

   Atrybut **TrigData** może zawierać nazwę kanału, który ma zostać wyzwolony, lub może on pozostać pusty. Jeśli pole IBM MQ for z/OSpozostanie puste, inicjator kanału przeszukuje pliki definicji kanału, dopóki nie znajdzie kanału, który jest powiązany z nazwaną kolejką transmisji. Gdy menedżer kolejek tworzy komunikat wyzwalacza, kopiuje on informacje z atrybutu **TrigData** definicji kolejki transmisji do komunikatu wyzwalacza.

5. Jeśli obiekt definicji procesu został utworzony w celu określenia właściwości aplikacji, która ma obsługiwać kolejkę aplikacji, powiąże obiekt procesu z kolejką aplikacji, nadając mu nazwę w atrybucie **ProcessName** w kolejce.

Platforma	Użyj komend
UNIX, Linux, and Windows systemy	ALTER QLOCAL
z/OS z/OS z/OS	ALTER QLOCAL
IBM i IBM i IBM i	CHGMQMQ

6. Uruchomienie instancji monitorów wyzwalacza IBM i (lub wyzwalania serwerów w programie IBM MQ for IBM i) , które mają służyć kolejkom inicjujowym, które zostały zdefiniowane. Więcej informacji zawiera sekcja “Przetwarzanie kolejki inicjuj przez monitory wyzwalacza” na stronie 973.

Aby uzyskać informacje o wszystkich niedostarczonych komunikatach wyzwalających, należy upewnić się, że menedżer kolejek ma zdefiniowaną kolejkę niedostarczonych komunikatów (niedostarczonych komunikatów). Podaj nazwę kolejki w polu menedżera kolejek produktu *DeadLetterQName* .

Następnie można ustawić wymagane warunki wyzwalacza, korzystając z atrybutów obiektu kolejki, które definiują kolejkę aplikacji. Więcej informacji na ten temat zawiera sekcja “Sterowanie zdarzeniami wyzwalającymi” na stronie 969.

Warunki dla zdarzenia wyzwalającego

Menedżer kolejek tworzy komunikat wyzwalacza, gdy spełnione są warunki określone w tym temacie.

Odwołania do kolejek współużytkowanych w tym temacie oznaczają kolejki współużytkowane w grupie współużytkowania kolejek, które są dostępne tylko w produkcie IBM MQ for z/OS.

Następujące warunki powodują, że menedżer kolejek tworzy komunikat wyzwalacza:

1. Komunikat jest *umieszczany* w kolejce.
2. Priorytet komunikatu ma priorytet większy niż lub równy progowi, który jest priorytetem kolejki. Ten priorytet jest ustawiany w atrybucie kolejki lokalnej **TriggerMsgPriority** ; jeśli jest ustawiony na zero, każdy komunikat kwalifikuje się do jakości.
3. Liczba komunikatów w kolejce z priorytetem większym lub równym *TriggerMsgPriority* była poprzednio, w zależności od *TriggerType*:
   - Zero (dla wyzwalacza typu MQTT_FIRST)
   - Dowolna liczba (dla wyzwalacza typu MQTT_EVERY)
   - *TriggerDepth* minus 1 (dla typu wyzwalacza MQTT_DEPTH)

Uwaga:

a. W przypadku niewspółużytkowanych kolejek lokalnych menedżer kolejek zlicza zarówno zatwierdzone, jak i niezatwierdzone komunikaty, gdy ocenia, czy istnieją warunki dla zdarzenia wyzwalającego. W rezultacie 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 odpowiednim produktem *WaitInterval*, tak aby aplikacja oczekiwała na dotarcie do niego komunikatów.

b. W przypadku lokalnych kolejek współużytkowanych menedżer kolejek liczy tylko zatwierdzone komunikaty.

4. Dla wyzwalania typu FIRST lub DEPTH, żaden program nie ma otwartej kolejki aplikacji na potrzeby usuwania komunikatów (oznacza to, że atrybut kolejki lokalnej **OpenInputCount** jest równy zero).

   Uwaga:

   a. W przypadku kolejek współużytkowanych warunki specjalne mają zastosowanie, gdy wiele menedżerów kolejek wyzwala monitory działające względem kolejki. W takiej sytuacji, jeśli co najmniej jeden menedżer kolejek ma otwartą kolejkę dla danych wejściowych współużytkowanych, kryteria wyzwalacza dla pozostałych menedżerów kolejek są traktowane jako *TriggerType* MQTT_FIRST i *TriggerMsgPriority* zero. Gdy wszystkie menedżery kolejek zamkną kolejkę dla danych wejściowych, warunki wyzwalacza są przywracane do warunków określonych w definicji kolejki.

      Przykładowym scenariuszem, na który wpływa ten warunek, jest wiele menedżerów kolejek QM1, QM2i QM3 z monitorem wyzwalacza uruchomionym dla kolejki aplikacji A. Komunikat dociera do A spełniającego warunki wyzwalania, a w kolejce inicjacji generowany jest komunikat wyzwalacza. Monitor wyzwalacza w menedżerze kolejek QM1 pobiera komunikat wyzwalacza i wyzwala aplikację. Wyzwalana aplikacja otwiera kolejkę aplikacji dla współużytkowanych danych wejściowych. Od tego momentu w przypadku warunków wyzwalacza dla kolejki aplikacji A są one oceniane jako *TriggerType* MQTT_FIRST, a *TriggerMsgPriority* zero w menedżerach kolejek QM2 i QM3, dopóki QM1 nie zamknie kolejki aplikacji.

   b. W przypadku kolejek współużytkowanych ten warunek jest stosowany dla każdego menedżera kolejek. Oznacza to, że *OpenInputCount* dla kolejki menedżera kolejek musi być zerem dla komunikatu wyzwalacza, który ma zostać wygenerowany dla kolejki przez tego menedżera kolejek. Jeśli jednak dowolny menedżer kolejek w grupie współużytkowania kolejek ma otwartą kolejkę za pomocą opcji MQOO_INPUT_EXCLUSIVE, dla tej kolejki nie zostanie wygenerowany żaden komunikat wyzwalacza przez żaden z menedżerów kolejek w grupie współużytkowania kolejek.

      Zmiana sposobu wartościowania warunków wyzwalacza jest wykonywana, gdy wyzwalana aplikacja otwiera kolejkę dla danych wejściowych. W scenariuszach, w których działa tylko jeden monitor wyzwalacza, inne aplikacje mogą mieć ten sam efekt, ponieważ w podobny sposób otwierają kolejkę aplikacji na potrzeby wprowadzania danych. Nie ma znaczenia, czy kolejka aplikacji została otwarta przez aplikację, która jest uruchamiana przez monitor wyzwalacza, czy przez inną aplikację. Jest to fakt, że kolejka jest otwarta dla wejścia w innym menedżerze kolejek, który powoduje zmianę kryteriów wyzwalacza.

5. Jeśli w systemie IBM MQ for z/OSkolejka aplikacji jest jedną z atrybutem **Usage** o wartości MQUS_NORMAL, żądania pobierania dla niej nie są zablokowane (to znaczy atrybut kolejki **InhibitGet** to MQQA_GET_ALLOWED). Ponadto, jeśli wyzwalana kolejka aplikacji jest kolejką z atrybutem **Usage** o wartości MQUS_XMITQ, żądania pobrania dla niej nie są blokowane.
6. Albo:
   - Atrybut kolejki lokalnej **ProcessName** dla kolejki 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 tym przypadku komunikat wyzwalacza zawiera atrybuty z następującymi wartościami:
     – **QName**: nazwa kolejki

- **ProcessName**: odstępy
- **TriggerData**: wyzwalanie danych
- **ApplType**: MQAT_UNKNOWN
- **ApplId**: odstępy
- **EnvData**: odstępy
- **UserData**: odstępy

7. Utworzono kolejkę inicjujową i określono ją w atrybucie kolejki lokalnej **InitiationQName** . Ponadto:
   - Żądania pobierania nie są blokowane dla kolejki inicjuj (to znaczy wartość atrybutu kolejki **InhibitGet** to MQQA_GET_ALLOWED).
   - Żądania umieszczania nie mogą być blokowane dla kolejki inicjuj (to znaczy, że wartością atrybutu kolejki **InhibitPut** musi być MQQA_PUT_ALLOWED).
   - Wartość atrybutu **Usage** kolejki inicjuj. musi mieć wartość MQUS_NORMAL.
   - W środowiskach, w których obsługiwane są kolejki dynamiczne, kolejka inicjujący nie może być kolejką dynamiczną, która została oznaczona jako logicznie usunięta.
8. Monitor wyzwalacza aktualnie ma otwartą kolejkę inicjują do usuwania komunikatów (oznacza to, że 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. Aby to zrobić, należy ustawić atrybut **trigger** podczas definiowania kolejki lub użyć komendy ALTER QLOCAL.
10. Typ wyzwalacza (atrybut kolejki lokalnej**TriggerType**) nie jest typu MQTT_NONE.

    Jeśli wszystkie wymagane warunki są spełnione, a komunikat, który spowodował warunek wyzwalacza, jest umieszczany jako część jednostki pracy, komunikat wyzwalacza nie będzie dostępny do pobrania przez aplikację monitora wyzwalacza do momentu zakończenia jednostki pracy, niezależnie od tego, czy jednostka pracy została zatwierdzona, czy też dla typu wyzwalacza MQTT_FIRST lub MQTT_DEPTH, dla którego utworzono kopię zapasową.
11. Odpowiedni komunikat jest umieszczany w kolejce, dla elementu **TriggerType** o wartości MQTT_FIRST lub MQTT_DEPTH, a także w kolejce:
    - Wcześniej nie było puste (MQTT_FIRST), lub
    - W przypadku produktu **TriggerDepth** lub większej liczby komunikatów (MQTT_DEPTH)

    i warunki od “2” na stronie 965 do “10” na stronie 967 (z wyjątkiem “3” na stronie 965) są spełnione, jeśli w przypadku MQTT_FIRST upłynęło wystarczająco dużo czasu (atrybut menedżera kolejek**TriggerInterval**), który upłynął od ostatniego komunikatu wyzwalacza, który został zapisany dla tej kolejki.

    Ma to na celu umożliwienie serwerowi kolejek, który kończy się przed przetworami wszystkich komunikatów w kolejce. Celem odstępu czasu wyzwalacza jest zmniejszenie liczby wygenerowanych duplikatów komunikatów wyzwalacza.

    Uwaga: Jeśli menedżer kolejek zostanie zatrzymany i zrestartowany, licznik czasu **TriggerInterval** zostanie zresetowany. Istnieje małe okno, podczas którego możliwe jest utworzenie dwóch komunikatów wyzwalacza. Okno istnieje, gdy atrybut wyzwalacza kolejki jest ustawiony na włączony w tym samym czasie co komunikat, a kolejka nie była wcześniej pusta (MQTT_FIRST) lub miała **TriggerDepth** lub więcej komunikatów (MQTT_DEPTH).
12. Jedyna aplikacja obsługując kolejkę wywołuje wywołanie MQCLOSE dla partycji **TriggerType** z MQTT_FIRST lub MQTT_DEPTH, a ponadto istnieje co najmniej:
    - Jedna (MQTT_FIRST), lub
    - **TriggerDepth** (MQTT_DEPTH)

    Komunikaty w kolejce o odpowiednim priorytecie (warunek “2” na stronie 965) i warunki “6” na stronie 966 za pomocą “10” na stronie 967 są również spełnione.

Ma to na celu umożliwienie serwerowi kolejek, który zgłasza wywołanie MQGET, opróżnia kolejkę i tak się kończy. Jednak w przedziale czasu między wywołaniami MQGET i MQCLOSE należy dotrzeć do jednego lub większej liczby komunikatów.

Uwaga:

a. Jeśli program obsługujący kolejkę aplikacji nie pobierze wszystkich komunikatów, może to spowodować pętlę zamkniętą. Za każdym razem, gdy program zamknie kolejkę, menedżer kolejek tworzy kolejny komunikat wyzwalacza, który powoduje, że monitor wyzwalacza ponownie uruchomi program serwera.

b. Jeśli program obsługujący kolejkę aplikacji wycofał żądanie pobrania (lub jeśli program się przerwie) przed zamknięciem kolejki, to samo dzieje się. Jeśli jednak program zamknie kolejkę przed utworzeniem kopii zapasowej żądania pobrania, a kolejka jest pusta, nie zostanie utworzony żaden komunikat wyzwalacza.

c. Aby zapobiec występowaniu takiej pętli, należy użyć pola *BackoutCount* deskryptora MQMD w celu wykrycia komunikatów, które są wielokrotnie wycofane. Więcej informacji na ten temat zawiera sekcja “Komunikaty, których kopie zapasowe zostały wycofane” na stronie 46.

13. Przy użyciu komendy MQSET lub komendy spełnione są następujące warunki:

a. • **TriggerControl** zmieniono na MQTC_ON, lub

• **TriggerControl** jest już MQTC_ON, a wartość **TriggerType**, **TriggerMsgPriority**lub **TriggerDepth** (jeśli jest odpowiednia) została zmieniona,

i jest co najmniej:

• Jedna (MQTT_FIRST lub MQTT_EVERY), lub

• **TriggerDepth** (MQTT_DEPTH)

komunikaty w kolejce o odpowiednim priorytecie (warunek “2” na stronie 965) oraz warunki “4” na stronie 966 za pomocą “10” na stronie 967 (z wyjątkiem “8” na stronie 967) są również zadowolone.

Jest to możliwe, aby aplikacja lub operator zmieniał kryteria wyzwalania, gdy warunki dla wyzwalacza do wystąpienia są już spełnione.

b. Wartość atrybutu kolejki **InhibitPut** w kolejce inicjuj zmienia się z MQQA_PUT_INHIBITED na MQQA_PUT_ALLOWED, a jest co najmniej:

• Jedna (MQTT_FIRST lub MQTT_EVERY), lub

• **TriggerDepth** (MQTT_DEPTH)

komunikaty o odpowiednim priorytecie (warunek “2” na stronie 965) w przypadku dowolnej kolejki, dla której jest to kolejka inicjujący, a warunki od “4” na stronie 966 do “10” na stronie 967 są również spełnione. (Jeden komunikat wyzwalacza jest generowany dla każdej takiej kolejki spełniający warunki.)

Polega to na tym, że komunikaty wyzwalacza nie są generowane z powodu warunku MQQA_PUT_INHIBITED w kolejce inicjuj, ale ten warunek został zmieniony.

c. Wartość atrybutu kolejki **InhibitGet** w kolejce aplikacji zmienia się z MQQA_GET_INHIBITED na MQQA_GET_ALLOWED, a istnieje co najmniej:

• Jedna (MQTT_FIRST lub MQTT_EVERY), lub

• **TriggerDepth** (MQTT_DEPTH)

komunikaty o odpowiednim priorytecie (warunek “2” na stronie 965) w kolejce, a warunki “4” na stronie 966 za pomocą “10” na stronie 967, z wyjątkiem “5” na stronie 966, są również spełnione.

Dzięki temu aplikacje mogą być wyzwalane tylko wtedy, gdy będą mogły pobierać komunikaty z kolejki aplikacji.

d. Aplikacja wyzwalacza-monitor wywołuje wywołanie MQOPEN dla danych wejściowych z kolejki inicjuj-i jest to co najmniej:

- Jedna (MQTT_FIRST lub MQTT_EVERY), lub
- **TriggerDepth** (MQTT_DEPTH)

komunikaty o odpowiednim priorytecie (warunek “2” na stronie 965) w przypadku dowolnej kolejki aplikacji, dla której jest to kolejka inicjujący, oraz warunki od “4” na stronie 966 do “10” na stronie 967 (z wyjątkiem “8” na stronie 967) są również usatysfakcjonowane, a żadna inna aplikacja nie ma otwartej kolejki inicjuj. dla danych wejściowych (generowany jest jeden komunikat wyzwalacza dla każdej takiej kolejki spełniający te warunki).

Ma to na celu umożliwienie komunikatów przychodzących do kolejek, gdy monitor wyzwalacza nie jest uruchomiony, a dla menedżera kolejek restartowanie i wyzwalanie komunikatów (które są nietrwałe) są tracone.

14. Parametr MSGDLVSQ jest ustawiony poprawnie. Jeśli parametr MSGDLVSQ=FIFO zostanie ustawiony, komunikaty będą dostarczane do kolejki w pierwszej pierwszej bazie danych. Priorytet komunikatu jest ignorowany, a domyślny priorytet kolejki jest przypisywany do komunikatu. Jeśli wartość **TriggerMsgPriority** jest ustawiona na wartość wyższą niż domyślny priorytet kolejki, komunikaty nie są wyzwalane. Jeśli parametr **TriggerMsgPriority** jest ustawiony na wartość równą lub niższą niż domyślny priorytet kolejki, wyzwalanie odbywa się dla typu FIRST, EVERY i DEPTH. Informacje na temat tych typów można znaleźć w opisie pola **TriggerType** w sekcji “Sterowanie zdarzeniami wyzwalającymi” na stronie 969.

    Jeśli parametr MSGDLVSQ=PRIORYTET zostanie ustawiony, a priorytet komunikatu jest równy lub większy niż pole *TriggerMsgPriority* , komunikaty będą liczone tylko w kierunku zdarzenia wyzwalającego. W tym przypadku wyzwalanie odbywa się dla typu FIRST, EVERY i DEPTH. Na przykład: jeśli zostanie wstawione 100 komunikatów o niższym priorytecie niż **TriggerMsgPriority**, efektywna głębokość kolejki dla celów wyzwalających będzie nadal zerowa. Jeśli następnie w kolejce zostanie wstawiony inny komunikat, ale tym razem priorytet jest większy lub równy **TriggerMsgPriority**, efektywna głębokość kolejki zwiększa się z zera do jednego, a warunek dla **TriggerType** FIRST jest spełniony.

Uwagi:

1. W kroku “12” na stronie 967 (w przypadku gdy komunikaty wyzwalacza są generowane w wyniku zdarzenia innego niż komunikat przylot do kolejki aplikacji), komunikat wyzwalacza nie jest umieszczany w jednostce pracy. Ponadto, jeśli parametr **TriggerType** ma wartość MQTT_EVERY, a w kolejce aplikacji występuje co najmniej jeden komunikat, generowany jest tylko jeden komunikat wyzwalacza.
2. Jeśli program IBM MQ segmuje komunikat podczas operacji MQPUT, zdarzenie wyzwalające nie będzie przetwarzane, dopóki wszystkie segmenty nie zostaną pomyślnie umieszczone w kolejce. Jednak po tym, jak segmenty komunikatów znajdują się w kolejce, program IBM MQ traktuje je jako pojedyncze komunikaty w celach wyzwalających. Na przykład pojedynczy komunikat logiczny podzielony na trzy części powoduje przetwarzanie tylko jednego zdarzenia wyzwalającego, gdy jest to pierwsza operacja MQPUT i segmentowana. Jednak każdy z tych trzech segmentów powoduje, że ich własne zdarzenia wyzwalające są przetwarzane w miarę ich przenoszenia przez sieć IBM MQ .
3. W przypadku systemu IBM MQ for z/OS, jeśli kolejka współużytkowana jest skonfigurowana do wyzwalania i połączenia z narzędziem CF udostępniającym kolejkę współużytkowaną, zdarzenie wyzwalające może zostać wygenerowane, a komunikat umieszczony w kolejce inicjujących. Może się to zdarzyć nawet wtedy, gdy do oryginalnej konfiguracji kolejki współużytkowanej nie został wyświetlony żaden komunikat. Jest to spowodowane nadmiernym oznaczeniem bitów przez makro IXLVECTR zgodnie z opisem w sekcji Wektor powiadomień o liście.

Sterowanie zdarzeniami wyzwalającymi

Zdarzenia wyzwalające sterują za pomocą niektórych atrybutów, które definiują kolejkę aplikacji. Informacje te zawierają również przykłady użycia typów wyzwalaczy: EVERY, FIRST i DEPTH.

Można włączyć i wyłączyć wyzwalanie, a także wybrać liczbę lub priorytet komunikatów, które są liczone w kierunku zdarzenia wyzwalającego. W sekcji Atrybuty obiektówznajduje się pełny opis tych atrybutów.

Odpowiednie atrybuty są następujące:

TriggerControl
Ten atrybut służy do włączania i wyłączania wyzwalania dla kolejki aplikacji.

TriggerMsgPriority
Minimalny priorytet, jaki musi być dla niego komunikat, aby był on liczony w kierunku zdarzenia wyzwalającego. Jeśli komunikat o priorytecie mniejszym niż *TriggerMsgPriority* dociera do kolejki aplikacji, menedżer kolejek zignoruje komunikat, gdy określi, czy ma zostać utworzony komunikat wyzwalacza. Jeśli wartość *TriggerMsgPriority* jest ustawiona na zero, wszystkie komunikaty są liczone w kierunku zdarzenia wyzwalającego.

TriggerType
Oprócz typu wyzwalacza NONE (który wyłącza wyzwalanie tak, jak ustawienie opcji *TriggerControl* na OFF), można użyć następujących typów wyzwalaczy, aby ustawić czułość kolejki w celu wyzwolenia zdarzeń:

Każdy
Zdarzenie wyzwalające występuje za każdym razem, gdy komunikat pojawia się w kolejce aplikacji. Tego typu wyzwalacza należy użyć w przypadku, gdy uruchomiono wiele instancji aplikacji.

pierwsza
Zdarzenie wyzwalające występuje tylko wtedy, gdy liczba komunikatów w kolejce aplikacji zostanie zmieniona z zera na jeden. Użyj tego typu wyzwalacza, jeśli chcesz, aby program obsługi był uruchamiany po nadejściu pierwszego komunikatu do kolejki, kontynuuj do momentu, aż nie będzie już więcej komunikatów do przetworzenia, a następnie zakończysz. Kolejka musi być zawsze przetwarzana, dopóki nie będzie pusta. Patrz także "Szczególny przypadek typu wyzwalacza FIRST" na stronie 971.

Głębokość
Zdarzenie wyzwalające występuje tylko wtedy, gdy liczba komunikatów w kolejce aplikacji osiągnie wartość atrybutu **TriggerDepth** . Typowym zastosowaniem tego typu wyzwalania jest uruchomienie programu, gdy wszystkie odpowiedzi na zestaw żądań zostaną odebrane.

Wyzwalanie według głębokości: W przypadku wyzwalania przez głębokość menedżer kolejek wyłącza wyzwalanie (przy użyciu atrybutu *TriggerControl*) po utworzeniu komunikatu wyzwalacza. Aplikacja musi ponownie włączyć wyzwalanie samego siebie (za pomocą wywołania MQSET) po tym, jak to się stało.

Działanie wyłączenia wyzwalania nie jest wykonywane w ramach elementu sterującego punktu synchronizacji, dlatego nie można ponownie włączyć wyzwalania, przy użyciu jednostki pracy. Jeśli program wycofuje żądanie umieszczenia, które spowodowało zdarzenie wyzwalające, lub jeśli program się przerwie, należy ponownie włączyć wyzwalanie za pomocą wywołania MQSET lub komendy ALTER QLOCAL.

TriggerDepth
Liczba komunikatów w kolejce, które powodują zdarzenie wyzwalające, gdy jest używane wyzwalanie przez głębokość.

Warunki, które muszą być spełnione dla menedżera kolejek w celu utworzenia komunikatu wyzwalacza, są opisane w sekcji "Warunki dla zdarzenia wyzwalającego" na stronie 965.

Przykład użycia wyzwalacza typu BARDZO

Rozważ zastosowanie aplikacji, która generuje żądania dotyczące ubezpieczenia ruchowego. Aplikacja może wysyłać komunikaty żądań do wielu towarzystw ubezpieczeniowych, podając tę samą kolejkę odpowiedzi do kolejki za każdym razem. Może on ustawić wyzwalacz typu EVERY w tej kolejce odpowiedzi, tak aby za każdym razem, gdy nadejdzie odpowiedź, odpowiedź może wyzwoliwać instancję serwera w celu przetworzenia odpowiedzi.

Przykład użycia typu wyzwalacza FIRST

Należy wziąć pod uwagę organizację z wieloma biurami oddziałów, z których każda przekazuje szczegóły dotyczące dni pracy do centrali. Wszyscy robią to jednocześnie, pod koniec dnia pracy, a w centrali

znajduje się aplikacja, która przetwarza detale ze wszystkich oddziałów oddziału. Pierwszy komunikat, który ma zostać dostarczony do centrali, może spowodować zdarzenie wyzwalające, które uruchamia tę aplikację. Ta aplikacja będzie kontynuować przetwarzanie, dopóki nie będzie więcej komunikatów znajdujących się w kolejce.

Przykład użycia wyzwalacza typu DEPTH

Należy rozważyć aplikację biura podróży, która tworzy pojedynczy wniosek o potwierdzenie rezerwacji lotu, potwierdzenie rezerwacji pokoju hotelowego, wynajem samochodu oraz zameldowanie niektórych podróżników. Aplikacja może rozdzielić te elementy na cztery komunikaty żądań, wysyłając każde z nich do osobnego miejsca docelowego. Może on ustawić wyzwalacz typu DEPTH w kolejce odpowiedzi do kolejki (z ustawioną głębią ustawioną na wartość 4), tak aby był on restartowany tylko wtedy, gdy wszystkie cztery odpowiedzi zostały nadesłane.

Jeśli w kolejce odpowiedzi przed ostatnią z czterech odpowiedzi nadchodzi inny komunikat (prawdopodobnie z innego żądania), aplikacja żądająca zostanie uruchomiona wcześniej. Aby tego uniknąć, podczas używania wyzwalacza DEPTH do gromadzenia wielu odpowiedzi na żądanie zawsze należy używać nowej kolejki odpowiedzi dla każdego żądania.

Szczególny przypadek typu wyzwalacza FIRST

W przypadku pierwszego typu wyzwalacza, jeśli w kolejce aplikacji istnieje już komunikat po nadejściu innego komunikatu, menedżer kolejek zwykle nie tworzy innego komunikatu wyzwalacza.

Jednak aplikacja obsługując kolejkę może w rzeczywistości nie otworzyć kolejki (na przykład może to zakończyć się aplikacją, prawdopodobnie z powodu problemu systemowego). Jeśli do obiektu definicji procesu została wstawiona niepoprawna nazwa aplikacji, aplikacja obsługując tę kolejkę nie odbierze żadnego z komunikatów. W takich sytuacjach, jeśli w kolejce aplikacji pojawi się inny komunikat, nie jest uruchomiony żaden serwer, który może przetworzyć ten komunikat (oraz inne komunikaty w kolejce).

Aby rozwiązać ten problem, menedżer kolejek tworzy kolejne komunikaty wyzwalacza w następujących okolicznościach:

- Jeśli w kolejce aplikacji pojawi się inny komunikat, ale tylko wtedy, gdy upłynął predefiniowany odstęp czasu od momentu utworzenia przez menedżer kolejek ostatniego komunikatu wyzwalacza dla tej kolejki. Ten przedział czasu jest zdefiniowany w atrybucie *TriggerInterval*menedżera kolejek. Jego wartość domyślna to 999 999 999 milisekund.
- W systemie IBM MQ for z/OSkolejki aplikacji, których nazwa jest nazwą otwartej kolejki inicjuj, są okresowo skanowane. Jeśli od momentu wysłania ostatniego komunikatu wyzwalacza upłynono *TRIGINT* milisekund, a kolejka spełnia warunki dla zdarzenia wyzwalającego, a wartość CURDEPTH jest większa od zera, generowany jest komunikat wyzwalacza. Ten proces jest nazywany wyzwalaniem backstop.

Podczas podejmowania decyzji o wartości dla odstępu czasu wyzwalacza, który ma być używany w aplikacji, należy wziąć pod uwagę następujące kwestie:

- Jeśli wartość *TriggerInterval* jest ustawiona na niską wartość i nie ma aplikacji obsługując kolejkę aplikacji, wyzwalacz typu FIRST może zachowywać się jak typ wyzwalacza EVERY. Zależy to od szybkości umieszczania komunikatów w kolejce aplikacji, która z kolei może zależeć od innych działań systemu. Wynika to z faktu, że jeśli przedział czasu wyzwalacza jest bardzo mały, za każdym razem, gdy komunikat jest umieszczany w kolejce aplikacji, generowany jest inny komunikat wyzwalacza, nawet jeśli typ wyzwalacza jest typu FIRST, a nie EVERY. (Typ wyzwalacza FIRST z odstępem czasu wyzwalacza równy zero jest równoznaczny z typem wyzwalacza EVERY.)
- W systemie IBM MQ for z/OS , jeśli parametr *TRIGINT* jest ustawiony na niską wartość i nie ma aplikacji obsługujących kolejkę aplikacji FIRST typu wyzwalacza, wyzwalanie backstop wygeneruje komunikat wyzwalający za każdym razem, gdy następuje okresowe skanowanie kolejek aplikacji o nazwach otwartych kolejek inicjuj.
- Jeśli tworzona jest kopia zapasowa jednostki pracy (patrz sekcja Wyzwalanie komunikatów i jednostek pracy) i interwał wyzwalacza został ustawiony na wysoką wartość (lub wartość domyślną), jeden komunikat wyzwalacza jest generowany, gdy jednostka pracy jest wycofana. Jeśli jednak odstęp

czasu wyzwalacza został ustawiony na niską wartość lub wartość zero (powoduje, że wyzwalacz typu FIRST to zachowuje się jak typ wyzwalacza EVERY), może zostać wygenerowany wiele komunikatów wyzwalacza. Jeśli kopia zapasowa jednostki pracy jest wycofana, wszystkie komunikaty wyzwalacza są nadal dostępne. Liczba komunikatów wyzwalacza, które są generowane, zależy od przedziału czasu wyzwalacza. Jeśli przedział czasu wyzwalacza jest ustawiony na zero, to generowana jest maksymalna liczba komunikatów.

Projektowanie aplikacji, która korzysta z wyzwalanych kolejek

Przekonałeś się, jak skonfigurować i kontrolować, wyzwalając dla swoich aplikacji. Poniżej znajdują się wskazówki, które należy wziąć pod uwagę podczas projektowania aplikacji.

Komunikaty wyzwalacza i jednostki pracy

Komunikaty wyzwalacza utworzone ze względu na zdarzenia wyzwalające, które nie są częścią jednostki pracy, są umieszczane w kolejce inicjujący, poza dowolną jednostką pracy, bez zależności od innych komunikatów i są dostępne do pobrania przez monitor wyzwalacza natychmiast.

Komunikaty wyzwalacza utworzone ze względu na zdarzenia wyzwalające, które są częścią jednostki pracy, są udostępniane w kolejce inicjuj, gdy jednostka pracy jest rozstrzygnięta, niezależnie od tego, czy jednostka pracy została zatwierdzona, czy wycofana

Jeśli menedżer kolejek nie umieje umieścić komunikatu wyzwalacza w kolejce inicjujący, zostanie on umieszczony w kolejce niedostarczonych komunikatów (niedostarczonych komunikatów).

Uwaga:

1. Menedżer kolejek liczy zarówno zatwierdzone, jak i niezatwierdzone komunikaty, gdy ocenia, 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 rozważmy żądanie umieszczenia w jednostce pracy dla kolejki, która jest wyzwalana z typem wyzwalacza FIRST. Powoduje to utworzenie komunikatu wyzwalacza przez menedżer kolejek. Jeśli wystąpi inne żądanie umieszczenia z innej jednostki pracy, nie powoduje to innego zdarzenia wyzwalającego, ponieważ liczba komunikatów w kolejce aplikacji została zmieniona z jednego na dwa, co nie spełnia warunków dla zdarzenia wyzwalającego. Jeśli zostanie utworzona kopia zapasowa pierwszej jednostki pracy, ale druga jest zatwierdzona, komunikat wyzwalacza jest nadal tworzony.

   Oznacza to jednak, że komunikaty wyzwalacza są czasami tworzone w sytuacji, gdy warunki dla zdarzenia wyzwalającego nie są spełnione. Aplikacje, które wykorzystują wyzwalanie, muszą być zawsze przygotowane do obsługi tej sytuacji. Zaleca się użycie opcji wait z wywołaniem MQGET, ustawiając odpowiednią wartość na *WaitInterval* .

   Utworzone komunikaty wyzwalacza są zawsze udostępniane, niezależnie od tego, czy jednostka pracy jest wycofana, czy zatwierdzana.
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, które korzystają z wyzwalania, należy pamiętać o tym, że może wystąpić opóźnienie między uruchomieniem programu monitorującego wyzwalacz a innymi komunikatami dostępnymi w kolejce aplikacji. Może się to zdarzyć, gdy komunikat, który powoduje, że zdarzenie wyzwalające jest zatwierdzane przed innymi.

Aby umożliwić dotarcie komunikatów, należy zawsze używać opcji wait, gdy używane jest wywołanie MQGET w celu usunięcia komunikatów z kolejki, dla której ustawione są warunki wyzwalacza. Wartość *WaitInterval* musi być wystarczająca, aby umożliwić najdłuższy rozsądny czas między umieszczonym komunikatem i zatwierdzonym wywołaniem. Jeśli komunikat jest wysyłany ze zdalnego menedżera kolejek, ten czas ma wpływ na:

- Liczba komunikatów, które zostały umieszczone przed zatwierdzoną
- Szybkość i dostępność łącza komunikacyjnego
- Wielkości komunikatów

Przykład sytuacji, w której należy użyć wywołania MQGET z opcją oczekiwania, należy wziąć pod uwagę ten sam przykład, który był używany podczas opisywania jednostek pracy. To było żądanie umieszczenia w jednostce pracy dla kolejki, która jest wyzwalana z typem wyzwalacza FIRST. To zdarzenie powoduje, że menedżer kolejek tworzy komunikat wyzwalacza. Jeśli wystąpi inne żądanie umieszczenia z innej jednostki pracy, nie powoduje to innego zdarzenia wyzwalającego, ponieważ liczba komunikatów w kolejce aplikacji nie została zmieniona z 0 na 1. Jeśli zostanie utworzona kopia zapasowa pierwszej jednostki pracy, ale druga jest zatwierdzona, komunikat wyzwalacza jest nadal tworzony. Dlatego komunikat wyzwalacza jest tworzony w czasie, gdy tworzona jest kopia zapasowa pierwszej jednostki pracy. Jeśli przed zatwierdzoną drugą wiadomą opóźnienie zostanie opóźnione, wyzwolona aplikacja może zaczekać na jego działanie.

W przypadku wyzwalania typu DEPTH, opóźnienie może wystąpić nawet wtedy, gdy wszystkie istotne komunikaty zostaną ostatecznie zatwierdzone. Załóżmy, że atrybut kolejki **TriggerDepth** ma wartość 2. Po pojawie się dwóch komunikatów w kolejce, drugi komunikat powoduje utworzenie komunikatu wyzwalacza. Jeśli jednak drugi komunikat jest pierwszym, który ma zostać zatwierdzony, jest w tym momencie dostępny komunikat wyzwalacza. Monitor wyzwalacza uruchamia program serwera, ale program może pobrać tylko drugi komunikat, dopóki nie zostanie zatwierdzony pierwszy. Dlatego program może wymagać oczekiwania na udostępnienie pierwszego komunikatu.

Należy zaprojektować aplikację tak, aby była ona przerwana, jeśli nie będą dostępne żadne komunikaty do pobrania, gdy upłynie okres oczekiwania. Jeśli jeden lub więcej komunikatów przybędzie później, należy użyć ponownie uruchamianej aplikacji w celu ich przetworzenia. Ta metoda uniemożliwia bezczynność aplikacji i niepotrzebnie korzystanie z zasobów.

Przetwarzanie kolejki inicjuj przez monitory wyzwalacza

Dla menedżera kolejek monitor wyzwalacza jest podobny do dowolnej innej aplikacji, która obsługuje kolejkę. Monitor wyzwalacza obsługuje jednak kolejki inicjujący.

Monitor wyzwalacza jest zwykle programem ciągłym. Po nadejściu komunikatu wyzwalacza w kolejce inicjujący monitor wyzwalacza pobiera ten komunikat. Używa on informacji w komunikacie do wydania komendy uruchamiających 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 poprawnej kolejce aplikacji.

Inicjator kanału jest przykładem specjalnego typu monitora wyzwalacza dla agentów kanałów komunikatów. Jednak w takiej sytuacji należy użyć albo wyzwalacza typu FIRST, albo DEPTH.

Windows UNIX *Monitory wyzwalacza w systemach UNIX i Windows*

Ten temat zawiera informacje na temat monitorów wyzwalaczy udostępnianych w systemach UNIX i Windows .

Dla środowiska serwera dostępne są następujące monitory wyzwalacza:

amqstrg0
: To jest przykładowy monitor wyzwalacza, który udostępnia podzbiór funkcji udostępnianej przez program **runmqtrm**. Więcej informacji na temat amqstrg0zawiera sekcja “Korzystanie z przykładowych programów na platformie Multiplatforms” na stronie 1166 .

runmqtrm
: Składnia tej komendy jest następująca: **runmqtrm** [*-m QMgrName*] [*-q InitQ*], gdzie QMgrName to menedżer kolejek, a InitQ to kolejka inicjujący. Domyślna kolejka 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 zbudowany w następujący sposób:

1. *ApplId* z odpowiedniej definicji PROCESS (jeśli została utworzona)
2. Struktura MQTMC2 ujęta w znaki podwójnego cudzysłowu.
3. *EnvData* z odpowiedniej definicji PROCESS (jeśli została utworzona)

gdzie *ApplId* to nazwa programu, który ma być uruchamiany w takiej postaci, w której jest wprowadzany w wierszu komend.

Przekazany parametr jest strukturą znaków MQTMC2 . Wywołano łańcuch komendy, który ma ten łańcuch, dokładnie tak, jak podano w podwójnych cudzysłowach, aby komenda systemowa zaakceptowała ją jako jeden parametr.

Monitor wyzwalacza nie widzi, czy istnieje inny komunikat w kolejce inicjujący do czasu zakończenia uruchomionego przez niego aplikacji. Jeśli aplikacja ma dużo przetwarzania, monitor wyzwalacza może nie być w stanie nadążać za liczbą przybywających komunikatów wyzwalacza. Dostępne są dwie opcje:

- Czy uruchomiono więcej monitorów wyzwalaczy
- Uruchom uruchomione aplikacje w tle

Jeśli uruchomionych jest więcej monitorów wyzwalacza, można sterować maksymalną liczbą aplikacji, które mogą być uruchamiane w dowolnym momencie. W przypadku uruchamiania aplikacji w tle nie ma ograniczeń narzuconych przez produkt IBM MQ w odniesieniu do liczby aplikacji, które mogą być uruchamiane.

Aby uruchomić uruchomionym aplikację w tle w systemach Windows , w polu *ApplId* , należy poprzedzić nazwę aplikacji komendą START. Na przykład:

```
START ?B AMQSECHA
```

UNIX Aby uruchomić uruchomionym aplikację w tle na serwerze UNIX, należy umieścić & na końcu *EnvData* definicji PROCESS.

Uwaga: Windows Jeśli ścieżka Windows zawiera spacje jako część nazwy ścieżki, należy ją ująć w cudzysłów ("). aby upewnić się, że jest ona obsługiwana jako pojedynczy argument. Na przykład "`C:\Program Files\Application Directory\Application.exe`".

Poniżej znajduje się przykład łańcucha APPLICID, w którym nazwa pliku zawiera spacje jako część ścieżki:

```
START "" /B "C:\Program Files\Application Directory\Application.exe"
```

Składnia komendy Windows START w przykładzie zawiera pusty łańcuch ujęty w podwójne cudzysłowy. Komenda START określa, że pierwszy argument w cudzysłowie będzie traktowany jako tytuł nowej komendy. Aby upewnić się, że program Windows nie pomylił ścieżki aplikacji dla argumentu 'title', dodaj do komendy łańcuch tytułowy ujęty w podwójny cudzysłów przed nazwą aplikacji.

Dla klienta IBM MQ dostępne są następujące monitory wyzwalacza:

runmqtmc
Jest to takie samo, jak runmqtrm, z wyjątkiem tego, że łączy się z bibliotekami produktu IBM MQ MQI client .

Windows UNIX *Monitor wyzwalacza dla CICS*

Monitor wyzwalacza amqltmc0 jest udostępniany dla produktu CICS. Działa on w taki sam sposób, jak standardowy monitor wyzwalacza, runmqtrm, ale uruchamiasz go w inny sposób i wyzwala transakcje CICS .

Ten temat dotyczy tylko systemów Windows, UNIXi Linux x86-64 .

Monitor wyzwalacza jest dostarczany jako program CICS ; zdefiniuj go przy użyciu 4-znakowej nazwy transakcji. Wprowadź 4-znakową nazwę, aby uruchomić monitor wyzwalacza. Używa on domyślnego menedżera kolejek (o nazwie określonej w pliku qm.ini lub w systemie IBM MQ for Windows, rejestru) oraz SYSTEM.CICS.INITIATION.QUEUE.

Jeśli ma być używany inny menedżer kolejek lub inny menedżer kolejek, należy zbudować strukturę monitora wyzwalacza MQTMC2 : wymaga to napisania programu za pomocą wywołania EXEC CICS START, ponieważ struktura jest zbyt długa, aby można ją było dodać jako parametr. Następnie przekaz strukturę MQTMC2 jako dane do żądania START dla monitora wyzwalacza.

Gdy używana jest struktura MQTMC2 , należy podać tylko parametry *StrucId*, *Version*, *QName*i **QMgrName** do monitora wyzwalacza, ponieważ nie odwołuje się do żadnych innych pól.

Komunikaty są odczytyane z kolejki inicjowania i używane do uruchamiania transakcji CICS przy użyciu komendy EXEC CICS START, przy założeniu, że parametr APPL_TYPE w komunikacie wyzwalacza ma wartość MQAT_CICS. Odczyt komunikatów z kolejki inicjujący jest wykonywany pod kontrolą punktu synchronizacji CICS .

Komunikaty są generowane, gdy monitor jest uruchamiany i zatrzymywane, a także w przypadku wystąpienia błędu. Komunikaty te są wysyłane do przejściowej kolejki danych CSMT.

Tabela 133. Dostępne wersje monitora wyzwalacza.

Tabela z dwiema kolumnami. pierwszy colum zawiera listę dostępnych wersji monitora wyzwalacza, a w drugiej kolumnie wyświetlane są platformy, dla których każda wersja jest używana.

Wersja	**Użyj**
amqltmc0	TXSeries Przez: • AIX AIX • Linux Systemy Linux x86-64 • Solaris Oracle Solaris wersja 5.1
amqltmc4	Windows TXSeries for Windows 5.1
amqltmcc	Wersja klienta powiązana z monitorem wyzwalacza CICS

Jeśli potrzebny jest monitor wyzwalacza dla innych środowisk, napisz program, który może przetwarzać komunikaty wyzwalacza, które menedżer kolejek umieszcza w kolejkach inicjujący. Taki program powinien wykonać następujące działania:

1. Użyj wywołania MQGET, aby zaczekać na pojaw się komunikatu w kolejce inicjuj.
2. Sprawdź pola struktury MQTM komunikatu wyzwalacza, aby znaleźć nazwę aplikacji, która ma zostać uruchomiona, oraz środowisko, w którym jest ona uruchamiana.
3. Wydaj komendę uruchomienia 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. Przekaz strukturę MQTMC2 lub MQTM do uruchomionej aplikacji. Może to zawierać dane użytkownika.

6. Powiąż z kolejką aplikacji aplikację, która ma obsługiwać tę kolejkę. W tym celu należy nadawać nazwę obiektowi definicji procesu (jeśli jest to utworzone) 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 można znaleźć w sekcji MQTMC2.

IBM i *Monitory wyzwalacza w systemie IBM i*
W systemie IBM i, zamiast komendy sterującej **runmqtrm** , należy użyć komendy CL IBM MQ for IBM i **STRMQMTRM**.

Użyj komendy STRMQMTRM w następujący sposób:

```
STRMQMTRM INITQNAME(InitQ) MQMNAME(QMgrName)
```

Szczegóły są następujące: runmqtrm.

Dostępne są również następujące programy przykładowe, które można wykorzystać jako modele 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 istnieje dodatkowe przetwarzanie powiązane z każdym komunikatem wyzwalacza.

AMQSERV4
Jest to serwer wyzwalacza. Dla każdego komunikatu wyzwalacza serwer ten uruchamia komendę dla procesu w swoim własnym zadaniu i może wywoływać transakcje CICS .

Zarówno monitor wyzwalacza, jak i serwer wyzwalacza przekazują strukturę MQTMC2 do programów, które są uruchamiane. Opis tej struktury można znaleźć w sekcji MQTMC2. Obie 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 wywoływać bezpośrednio programów Java , ponieważ klasy Java znajdują się w systemie plików IFS. Jednak programy Java mogą być wyzwalane pośrednio, wywołując program CL, który następnie wywołuje program Java i przechodzi przez strukturę TMC2 . Minimalna wielkość struktury TMC2 wynosi 732 bajty.

Poniżej znajduje się 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
```

Dostępny jest następujący program monitorujący wyzwalacza dla IBM MQ MQI client: 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 977
- “Komunikaty restartu i wyzwalacza menedżera kolejek” na stronie 977
- “Wyzwalanie komunikatów i wprowadzanie zmian w atrybutach obiektów” na stronie 977
- “Format komunikatów wyzwalacza” na stronie 977

Trwałość i priorytet komunikatów wyzwalacza

Komunikaty wyzwalacza nie są trwałe, ponieważ nie ma potrzeby, aby je tak było.

Jednak warunki generowania zdarzeń wyzwalających są trwałe, dlatego komunikaty wyzwalacza są generowane zawsze wtedy, gdy warunki te są spełnione. Jeśli komunikat wyzwalacza zostanie utracony, dalsze istnienie komunikatu aplikacji w kolejce aplikacji gwarantuje, że menedżer kolejek wygeneruje komunikat wyzwalający, gdy tylko spełnione zostaną wszystkie warunki.

Jeśli jednostka pracy jest wycofana, wszystkie wygenerowane przez niego komunikaty wyzwalacza są zawsze dostarczane.

Komunikaty wyzwalacza przyjmują domyślny priorytet kolejki inicjujący.

Komunikaty restartu i wyzwalacza menedżera kolejek

Po zrestartowaniu menedżera kolejek po kolejnym otwarciu kolejki inicjuj. dla danych wejściowych, komunikat wyzwalacza może zostać umieszczony w tej kolejce inicjuj, jeśli kolejka aplikacji powiązana z nią zawiera komunikaty i jest zdefiniowana dla wyzwalania.

Wyzwalanie komunikatów i wprowadzanie zmian w atrybutach obiektów

Komunikaty wyzwalacza są tworzone zgodnie z wartościami atrybutów wyzwalacza, które są używane w czasie zdarzenia wyzwalającego.

Jeśli komunikat wyzwalacza nie jest dostępny dla monitora wyzwalacza do czasu późniejszego (ponieważ komunikat, który spowodował jego wygenerowanie, został umieszczony w jednostce pracy), wszystkie zmiany atrybutów wyzwalacza w międzyczasie nie mają wpływu na komunikat wyzwalacza. W szczególności wyłączenie wyzwalania nie zapobiega udostępnianiu komunikatu wyzwalacza po jego utworzeniu. Ponadto kolejka aplikacji może nie istnieć w momencie, gdy komunikat wyzwalacza jest udostępniony.

Format komunikatów wyzwalacza

Format komunikatu wyzwalacza jest definiowany przez strukturę MQTM.

Ma ona następujące pola, które menedżer kolejek wypełnia podczas tworzenia komunikatu wyzwalacza, używając 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, wypełni to pole za pomocą atrybutu **QName** w kolejce aplikacji.

ProcessName
Nazwa obiektu definicji procesu, który jest powiązany z kolejką aplikacji. Gdy menedżer kolejek tworzy komunikat wyzwalacza, wypełni to pole za pomocą atrybutu **ProcessName** w kolejce aplikacji.

TriggerData
Pole w formacie wolnoformatowym używane przez monitor wyzwalacza. Gdy menedżer kolejek tworzy komunikat wyzwalacza, wypełni to pole za pomocą atrybutu **TriggerData** w kolejce aplikacji.
W przypadku dowolnego produktu IBM MQ z wyjątkiem IBM MQ for z/OS, pole to może być używane do określenia nazwy kanału, który ma zostać wyzwolony.

ApplType
Typ aplikacji, która ma zostać uruchomiona przez monitor wyzwalacza. Gdy menedżer kolejek tworzy komunikat wyzwalacza, wypełni to pole za pomocą atrybutu **ApplType** obiektu definicji procesu określonego w produkcie *ProcessName*.

ApplId
Łańcuch znaków identyfikujący aplikację, która ma zostać uruchomiona przez monitor wyzwalacza. Gdy menedżer kolejek tworzy komunikat wyzwalacza, wypełni to pole za pomocą atrybutu **ApplId** obiektu definicji procesu określonego w produkcie *ProcessName*.

Jeśli używany jest monitor wyzwalacza CKTI, dostarczony przez CICS, atrybut **ApplId** obiektu definicji procesu jest identyfikatorem transakcji CICS .

Jeśli używany jest CSQQTRMN dostarczony przez IBM MQ for z/OS, atrybut **ApplId** obiektu definicji procesu jest identyfikatorem transakcji IMS .

EnvData
Pole znakowe zawierające dane związane ze środowiskiem do użycia przez monitor wyzwalacza. Gdy menedżer kolejek tworzy komunikat wyzwalacza, wypełni to pole za pomocą atrybutu **EnvData** obiektu definicji procesu określonego w produkcie *ProcessName*. Monitor wyzwalacza dostarczony przez produkt CICS(CKTI) lub monitor wyzwalacza dostarczony przez produkt IBM MQ for z/OS(CSQQTRMN) nie korzystają z tego pola, ale inne monitory wyzwalacza mogą z niego korzystać.

UserData
Pole znakowe zawierające dane użytkownika przeznaczone do użycia przez monitor wyzwalacza. Gdy menedżer kolejek tworzy komunikat wyzwalacza, wypełni to pole za pomocą atrybutu **UserData** obiektu definicji procesu określonego w produkcie *ProcessName*. To pole może być użyte do określenia nazwy kanału, który ma zostać wyzwolony.

W programie MQTMznajduje się pełny opis struktury komunikatu wyzwalacza.

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 wartość applid w obiekcie procesu musi określać, że program ma być uruchomiony w tle. W przeciwnym razie monitor wyzwalacza nie może uruchomić programu.

Jeśli komunikat wyzwalacza został utworzony, ale nie można go umieścić w kolejce inicjuj (na przykład, ponieważ kolejka jest pełna lub długość komunikatu wyzwalacza jest większa niż maksymalna długość komunikatu określona dla kolejki inicjuj), komunikat wyzwalacza jest umieszczany w kolejce niedostarczonych komunikatów (niedostarczonych komunikatów).

Jeśli operacja put dla kolejki niedostarczonych komunikatów nie może zostać zakończona pomyślnie, komunikat wyzwalacza jest odrzucany, a komunikat ostrzegawczy jest wysyłany z/OS do konsoli z/OS lub do operatora systemu lub jest umieszczany w dzienniku błędów.

Umieszczenie komunikatu wyzwalacza w kolejce niedostarczonych komunikatów może spowodować wygenerowanie komunikatu wyzwalacza dla tej kolejki. Ten drugi komunikat wyzwalacza jest odrzucany, jeśli do kolejki niedostarczonych komunikatów zostanie dodany komunikat.

Jeśli program jest wyzwalany pomyślnie, ale kończy się przed odebraniem komunikatu z kolejki, należy użyć programu narzędziowego śledzenia (na przykład CICS AUXTRACE, jeśli program działa w ramach programu CICS). w celu znalezienia przyczyny niepowodzenia.

Praca z interfejsem MQI i klastrami

Istnieją specjalne opcje dotyczące wywołań i kodów powrotu, które odnoszą się do technologii klastrowej.

Użyj poniższych odsyłaczy, aby dowiedzieć się więcej na temat opcji dostępnych w wywołaniach i kodach powrotu do użycia z klastrami:

- “MQOPEN i klastry” na stronie 979
- “MQPUT, MQPUT1 i klastry” na stronie 980
- “MQINQ i klastry” na stronie 981
- “MQSET i klastry” na stronie 982
- “Kody powrotu” na stronie 982

Pojęcia pokrewne

“Interfejs kolejki komunikatów-przegląd” na stronie 811
Informacje na temat komponentów interfejsu kolejek komunikatów (Message Queue Interface-MQI).

“Nawiąże połączenie z menedżerem kolejek i odłączenie go od niego” na stronie 824
Aby można było korzystać z usług programistycznych produktu IBM MQ , program musi mieć połączenie z menedżerem kolejek. Informacje zawarte w tej sekcji umożliwiają zapoznanie się z informacjami na temat nawiązywania połączenia z menedżerem kolejek i z jego rozłączeniem.

“Otwieranie i zamykanie obiektów” na stronie 833
Te informacje udostępniają wgląd w otwieranie i zamykanie obiektów produktu IBM MQ .

“Umieszczanie komunikatów w kolejce” na stronie 844
Te informacje umożliwiają zapoznanie się z informacjami na temat umieszczania komunikatów w kolejce.

“Pobieranie komunikatów z kolejki” na stronie 860
Ta sekcja zawiera informacje na temat pobierania komunikatów z kolejki.

“Sprawdzanie i ustawianie atrybutów obiektu” na stronie 943
Atrybuty to właściwości, które definiują parametry obiektu IBM MQ .

“Zatwierdzanie i wycofywanie jednostek pracy” na stronie 946
W tej sekcji opisano sposób zatwierdzania i tworzenia kopii zapasowych wszystkich możliwych do odtworzenia operacji get i put, które wystąpiły w jednostce pracy.

“Uruchamianie aplikacji produktu IBM MQ przy użyciu wyzwalaczy” na stronie 958
Informacje na temat wyzwalaczy i sposobów uruchamiania aplikacji IBM MQ przy użyciu wyzwalaczy.

“Używanie i zapisywanie aplikacji w systemie IBM MQ for z/OS” na stronie 983
Aplikacje produktu IBM MQ for z/OS mogą być uruchamiane z programów działających w wielu różnych środowiskach. Oznacza to, że mogą skorzystać z udogodnień dostępnych w więcej niż jednym środowisku.

“Aplikacje pomostowe IMS i IMS w systemie IBM MQ for z/OS” na stronie 70
Te informacje ułatwiają pisanie aplikacji produktu IMS przy użyciu produktu IBM MQ.

MQOPEN i klastry

Kolejka, do której komunikat jest umieszczany lub odczytany, gdy kolejka klastra jest otwierana, zależy od wywołania MQOPEN .

Wybieranie kolejki docelowej

Jeśli w deskryptorze obiektu nie zostanie podana nazwa menedżera kolejek, MQOD, menedżer kolejek wybiera menedżera 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, jeśli istnieje kolejka lokalna. Jeśli istnieje lokalna instancja kolejki, jest ona zawsze otwierana w preferencjach do zdalnej instancji, chyba że atrybut CLWLUSEQ jest ustawiony na 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 zaokrąglany, korzystając z algorytmu zarządzania obciążeniem. Patrz sekcja Równoważenie obciążenia w klastrach.

Gdy używany jest algorytm równoważenia obciążenia, zależy od sposobu otwierania kolejki klastra:

- MQOO_BIND_ON_OPEN -algorytm jest używany raz w momencie otwierania kolejki przez aplikację.
- MQOO_BIND_NOT_FIXED -algorytm jest używany dla każdego komunikatu umieszczonego 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 umieszczone w kolejce, określające uchwyt obiektu zwróconego z wywołania MQOPEN , są kierowane do tego samego menedżera kolejek.

- Jeśli komunikaty mają powinowactwa, należy użyć opcji MQOO_BIND_ON_OPEN . Na przykład, jeśli zadanie wsadowe komunikatów ma być przetwarzane przez ten sam menedżer kolejek, należy określić wartość MQOO_BIND_ON_OPEN po otwarciu kolejki. Program IBM MQ naprawia menedżera kolejek i trasę, która ma być podejmowana przez wszystkie komunikaty umieszczone w tej kolejce.
- Jeśli podano opcję MQOO_BIND_ON_OPEN , należy ponownie otworzyć kolejkę dla nowej instancji kolejki, która ma zostać wybrana.

MQOO_BIND_NOT_FIXED
Opcja MQOO_BIND_NOT_FIXED w wywołaniu MQOPEN określa, że docelowy menedżer kolejek nie został naprawiony. Komunikaty zapisywane w kolejce, określające uchwyt obiektu zwróconego z wywołania MQOPEN , są kierowane do menedżera kolejek w czasie MQPUT na podstawie komunikatów typu message-by-message. Opcji MQOO_BIND_NOT_FIXED należy używać, jeśli nie ma potrzeby wymuszania zapisywania wszystkich komunikatów w tym samym miejscu docelowym.

- Nie należy jednocześnie podawać opcji MQOO_BIND_NOT_FIXED i MQMF_SEGMENTATION_ALLOWED . W takim przypadku segmenty komunikatu mogą być dostarczane do różnych menedżerów kolejek, rozrzuconych po całym klastrze.

MQOO_BIND_ON_GROUP
Umożliwia aplikacji żądanie, aby grupa komunikatów została przydzielona do tej samej instancji docelowej. Ta opcja jest poprawna tylko dla kolejek i dotyczy tylko kolejek klastra. Jeśli ta opcja jest określona dla kolejki, która nie jest kolejką klastra, opcja jest ignorowana.

- Grupy są kierowane do jednego miejsca docelowego tylko wtedy, gdy w tabeli MQPUT określono parametr MQPMO_LOGICAL_ORDER. Jeśli określono parametr MQOO_BIND_ON_GROUP, ale komunikat nie jest częścią grupy logicznej, zamiast niego używane jest zachowanie BIND_NOT_FIXED.

MQOO_BIND_AS_Q_DEF
Jeśli użytkownik nie poda opcji MQOO_BIND_ON_OPEN, MQOO_BIND_NOT_FIXED lub MQOO_BIND_ON_GROUP, domyślną opcją będzie MQOO_BIND_AS_Q_DEF. Użycie produktu MQOO_BIND_AS_Q_DEF powoduje, że powiązanie jest używane dla uchwytu kolejki, który ma zostać użyty z atrybutu kolejki DefBind .

Istotność opcji produktu MQOPEN

Opcje MQOPEN MQOO_BROWSE, MQOO_INPUT_*lub MQOO_SET wymagają, aby lokalna instancja kolejki klastra dla MQOPEN powiodła się.

The MQOPEN options MQOO_OUTPUT, MQOO_BIND_*, or MQOO_INQUIRE do not require a local instance of the cluster queue to succeed.

Rozstrzygnięta nazwa menedżera kolejek

Gdy nazwa menedżera kolejek jest rozstrzygana w czasie MQOPEN , rozstrzygnięta nazwa jest zwracana do aplikacji. Jeśli aplikacja podejmie próbę użycia tej nazwy w kolejnym wywołaniu programu MQOPEN , może się okazać, że nie ma uprawnień dostępu do nazwy.

MQPUT, MQPUT1 i klastry

Jeśli MQOO_BIND_NOT_FIXED jest określone na MQOPEN , procedury zarządzania obciążeniem wybierają miejsce docelowe MQPUT lub MQPUT1 .

Jeśli w wywołaniu MQOPEN określono wartość MQOO_BIND_NOT_FIXED , każde kolejne wywołanie programu MQPUT wywołuje procedurę zarządzania obciążeniem w celu określenia menedżera kolejek, do którego ma zostać wysłany komunikat. Miejsce docelowe i trasa do podjęcia są wybierane na podstawie komunikatu. Miejsce docelowe i trasa mogą ulec zmianie po umieszczniu komunikatu w razie zmiany

warunków w sieci. Wywołanie MQPUT1 zawsze działa tak, jakby MQOO_BIND_NOT_FIXED było aktywne, to znaczy zawsze wywołuje procedurę zarządzania obciążeniem.

Po wybraniu przez procedurę zarządzania obciążeniem menedżera kolejek lokalny menedżer kolejek kończy operację put. Komunikat może być umieszczony 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 tej samej nazwie, co docelowy menedżer kolejek.

Jeśli w wywołaniu MQOPEN określono wartość MQOO_BIND_ON_OPEN , wywołania MQPUT nie wywołują procedury zarządzania obciążeniem, ponieważ miejsce docelowe i trasa zostały już wybrane.

MQINQ i klastry

Wybór kolejki klastra zależy od opcji, które łączą się z produktem MQOO_INQUIRE.

Zanim możliwe będzie sprawdzenie kolejki, należy otworzyć ją za pomocą wywołania MQOPEN i określić MQOO_INQUIRE.

Aby uzyskać informacje na temat kolejki klastra, należy użyć wywołania MQOPEN i połączyć inne opcje z programem MQOO_INQUIRE. Atrybuty, które można sprawdzić, zależą od tego, czy istnieje lokalna instancja kolejki klastra, oraz od tego, w jaki sposób zostanie otwarta kolejka:

- Łączenie produktów MQOO_BROWSE, MQOO_INPUT_*lub MQOO_SET z produktem MQOO_INQUIRE wymaga lokalnej instancji kolejki klastra, aby możliwe było pomyślne wykonanie tej operacji. W tym przypadku można zapytać o wszystkie atrybuty, które są poprawne dla kolejek lokalnych.
- Łączenie produktu MQOO_OUTPUT z produktem MQOO_INQUIREi określenie żadnej z poprzednich opcji powoduje, że instancja otwarta jest:
  - Instancja w lokalnym menedżerze kolejek, jeśli istnieje. W tym przypadku można zapytać o wszystkie atrybuty, które są poprawne dla kolejek lokalnych.
  - Instancja w innym miejscu w klastrze, jeśli nie istnieje lokalna instancja menedżera kolejek. W tym przypadku można uzyskać dostęp tylko do następujących atrybutów. W tym przypadku atrybut QType ma wartość MQQT_CLUSTER .
    - DefBind
    - DefPersistence
    - DefPriority
    - InhibitPut
    - QDesc
    - Nazwa QName
    - QTYPE

Aby uzyskać informacje na temat atrybutu DefBind w kolejce klastra, należy użyć wywołania MQINQ z selektorem MQIA_DEF_BIND. Zwrócona wartość to MQBND_BIND_ON_OPEN lub MQBND_BIND_NOT_FIXEDalbo 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 konieczności naprawiać kolejki, z którą powiązana jest MQOPEN , kolejne wywołania programu MQINQ mogą zapytać o różne instancje kolejki klastra.

Pojęcia pokrewne

“Opcja MQOPEN dla kolejki klastra” na stronie 839

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 (opcja MQOO_SET) wymaga, aby lokalna instancja kolejki klastra dla MQSET powiodła się.

Nie można użyć wywołania MQSET do ustawienia atrybutów kolejki w innym miejscu w klastrze.

Można otworzyć lokalny alias lub kolejkę zdalną zdefiniowaną za pomocą atrybutu klastra, a następnie użyć wywołania MQSET . Istnieje możliwość ustawienia atrybutów 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')

Wywołano komendę MQOPEN, MQPUTlub MQPUT1 , aby otworzyć kolejkę klastra lub umieścić na niej komunikat. Wyjście obciążenia klastra, zdefiniowane za pomocą atrybutu ClusterWorkloadExit menedżera kolejek, nieoczekiwanie kończy się niepowodzeniem lub nie odpowiada w czasie.

W dzienniku systemowym w systemie IBM MQ for z/OS zapisywany jest komunikat zawierający więcej informacji o tym błędzie.

Kolejne wywołania funkcji 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.

Komunikat jest zapisywany w dzienniku systemowym, a przetwarzanie jest kontynuowane tak, jakby atrybut ClusterWorkloadExit był pusty.

Multi W systemie Wiele platformwysyłane jest 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 menedżera kolejek ClusterWorkloadExit menedżera kolejek.

MQRC_CLUSTER_PUT_INHIBITED (2268 X'8DC')

Wywołanie MQOPEN z opcjami MQOO_OUTPUT i MQOO_BIND_ON_OPEN w efekcie jest wydawane dla kolejki klastra. Wszystkie instancje kolejki w klastrze są obecnie wstrzymane przez ustawienie atrybutu InhibitPut ustawionego na wartość MQQA_PUT_INHIBITED. Ponieważ nie są dostępne żadne instancje kolejek do odbierania komunikatów, wywołanie MQOPEN nie powiodło się.

Ten kod przyczyny pojawia się tylko wtedy, gdy spełnione są oba poniższe instrukcje:

- Nie istnieje lokalna instancja kolejki. Jeśli istnieje instancja lokalna, wywołanie MQOPEN powiedzie się, nawet jeśli lokalna instancja jest zablokowana.
- Dla kolejki nie ma wyjścia obciążenia klastra lub jest wyjście obciążenia klastra, ale nie wybiera instancji kolejki. (Jeśli wyjście obciążenia klastra wybierze instancję kolejki, wywołanie MQOPEN zakończy się powodzeniem, nawet jeśli ta instancja jest zablokowana).

Jeśli w wywołaniu MQOPEN zostanie podana opcja MQOO_BIND_NOT_FIXED , wywołanie może zakończyć się powodzeniem, nawet jeśli wszystkie kolejki w klastrze zostaną zablokowane. Jednak kolejne wywołanie programu MQPUT może się nie powieść, jeśli wszystkie kolejki są nadal wstrzymane w czasie wywołania.

MQRC_CLUSTER_RESOLUTION_ERROR (2189 X'88D')

1. Wywołano komendę MQOPEN, MQPUTlub MQPUT1 , aby otworzyć kolejkę klastra lub umieścić na niej komunikat. Nie można poprawnie rozstrzygnąć definicji kolejki, ponieważ wymagana jest odpowiedź od menedżera kolejek pełnego repozytorium, ale żadna nie jest dostępna.

2. Wywołanie funkcji MQOPEN, MQPUT, MQPUT1 lub MQSUB jest wykonywane dla obiektu tematu z określeniem PUBSCOPE (ALL) lub SUBSCOPE (ALL). Nie można poprawnie rozstrzygnąć definicji tematu klastra, ponieważ wymagana jest odpowiedź z menedżera kolejek pełnego repozytorium, ale żadna z nich nie jest dostępna.

MQRC_CLUSTER_RESOURCE_ERROR (2269 X'8DD')

Wywołanie MQOPEN, MQPUTlub MQPUT1 jest wykonywane dla kolejki klastra. Wystąpił błąd podczas próby użycia zasobu wymaganego do łączenia w klastry.

MQRC_NO_DESTINATIONS_AVAILABLE (2270 X'8DE')

Wywołano komendę MQPUT lub MQPUT1 w celu umieszczenia komunikatu w kolejce klastra. W czasie wywołania nie ma już żadnych instancji kolejki w klastrze. MQPUT nie powiodło się, a komunikat nie został wysłany.

Błąd może wystąpić, jeśli w wywołaniu MQOPEN , która otwiera kolejkę, zostanie podana wartość MQOO_BIND_NOT_FIXED , lub MQPUT1 zostanie użyta do umieszczenia komunikatu.

MQRC_STOPPED_BY_CLUSTER_EXIT (2188 X'88C')

Wywołano komendę MQOPEN, MQPUTlub MQPUT1 , aby otworzyć lub umieścić komunikat w kolejce klastra. Wyjście obciążenia klastra odrzuca wywołanie.

z/OS Używanie i zapisywanie aplikacji w systemie IBM MQ for z/OS

Aplikacje produktu IBM MQ for z/OS mogą być uruchamiane z programów działających w wielu różnych środowiskach. Oznacza to, że mogą skorzystać 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 IBM MQ-CICS bridgezawiera sekcja Używanie produktu IBM MQ z produktem CICS .
- Informacje na temat korzystania z produktu IMS i mostu IMS można znaleźć w sekcji “Aplikacje pomostowe IMS i IMS w systemie IBM MQ for z/OS” na stronie 70.

Aby dowiedzieć się więcej na temat używania i zapisywania aplikacji w systemie IBM MQ for z/OS, należy użyć następujących odsyłaczy:

- “Funkcje IBM MQ for z/OS zależne od środowiska” na stronie 984
- “Narzędzia do debugowania, obsługa punktu synchronizacji i obsługa odtwarzania” na stronie 985
- “Interfejs IBM MQ for z/OS ze środowiskiem aplikacji” na stronie 985
- “Pisanie aplikacji z/OS UNIX System Services” na stronie 987
- “Programowanie aplikacji z kolejkami współużytkowanymi” na stronie 990

Pojęcia pokrewne

“Interfejs kolejki komunikatów-przegląd” na stronie 811
Informacje na temat komponentów interfejsu kolejek komunikatów (Message Queue Interface-MQI).

“Nawiąże połączenie z menedżerem kolejek i odłączenie go od niego” na stronie 824
Aby można było korzystać z usług programistycznych produktu IBM MQ , program musi mieć połączenie z menedżerem kolejek. Informacje zawarte w tej sekcji umożliwiają zapoznanie się z informacjami na temat nawiązywania połączenia z menedżerem kolejek i z jego rozłączeniem.

“Otwieranie i zamykanie obiektów” na stronie 833
Te informacje udostępniają wgląd w otwieranie i zamykanie obiektów produktu IBM MQ .

“Umieszczanie komunikatów w kolejce” na stronie 844
Te informacje umożliwiają zapoznanie się z informacjami na temat umieszczania komunikatów w kolejce.

“Pobieranie komunikatów z kolejki” na stronie 860
Ta sekcja zawiera informacje na temat pobierania komunikatów z kolejki.

“Sprawdzanie i ustawianie atrybutów obiektu” na stronie 943

Atrybuty to właściwości, które definiują parametry obiektu IBM MQ .

“Zatwierdzanie i wycofywanie jednostek pracy” na stronie 946
W tej sekcji opisano sposób zatwierdzania i tworzenia kopii zapasowych wszystkich możliwych do odtworzenia operacji get i put, które wystąpiły w jednostce pracy.

“Uruchamianie aplikacji produktu IBM MQ przy użyciu wyzwalaczy” na stronie 958
Informacje na temat wyzwalaczy i sposobów uruchamiania aplikacji IBM MQ przy użyciu wyzwalaczy.

“Praca z interfejsem MQI i klastrami” na stronie 978
Istnieją specjalne opcje dotyczące wywołań i kodów powrotu, które odnoszą się do technologii klastrowej.

“Aplikacje pomostowe IMS i IMS w systemie IBM MQ for z/OS” na stronie 70
Te informacje ułatwiają pisanie aplikacji produktu IMS przy użyciu produktu IBM MQ.

Funkcje IBM MQ for z/OS zależne od środowiska

Te informacje są używane podczas rozważania funkcji produktu IBM MQ for z/OS .

Główne różnice, które należy uwzględnić między funkcjami produktu IBM MQ w środowiskach, w których działa produkt IBM MQ for z/OS , są następujące:

- Produkt IBM MQ for z/OS udostępnia następujące monitory wyzwalacza:
  - CKTI do użycia w środowisku CICS
  - CSQQTRMN do użycia w środowisku IMS

  Aby uruchamiać aplikacje w innych środowiskach, należy napisać własny moduł.
- W środowiskach CICS i IMS obsługiwane jest synchronizowanie z użyciem zatwierdzania dwufazowego. Jest on również obsługiwany w środowisku wsadowym produktu z/OS przy użyciu zarządzania transakcjami i odtwarzalnych usług menedżera zasobów (RRS). Zatwierdzanie jednofazowe jest obsługiwane w środowisku z/OS przez samego IBM MQ .
- W środowiskach wsadowych i IMS interfejs MQI udostępnia wywołania służące do łączenia programów z menedżerem kolejek oraz ich odłączenia od menedżera kolejek. Programy mogą łączyć się z więcej niż jednym menedżerem kolejek.
- System CICS może łączyć się tylko z jednym menedżerem kolejek. Może się to zdarzyć, gdy produkt CICS zostanie zainicjowany, jeśli nazwa podsystemu jest zdefiniowana w zadaniu uruchamiania systemu CICS . Połączenia MQI i połączenia z rozłączeniem są tolerowane, ale nie mają wpływu, w środowisku CICS .
- Wyjście funkcji API pozwala na interwencję programu w przetwarzaniu wszystkich wywołań MQI. To wyjście jest dostępne tylko w środowisku CICS .
- W systemie CICS w systemach wieloprocesorowych, uzyskiana jest pewna zaleta wydajności, ponieważ wywołania MQI mogą być wykonywane w wielu obiektach TCB z/OS . Więcej informacji na ten temat zawiera podręcznik Planowanie w systemie z/OS *IBM MQ for z/OS Concepts and Planning Guide*.

Te funkcje są podsumowane w produkcie Tabela 134 na stronie 984.

Tabela 134. Opcje środowiskowe produktu z/OS

	CICS	**IMS**	**Batch/TSO**
Dostarczony 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 usługi RRS.

Narzędzia do debugowania, obsługa punktu synchronizacji i obsługa odtwarzania

Te informacje umożliwiają zapoznanie się z informacjami na temat narzędzi do debugowania programów, obsługi punktów synchronizacji i obsługi odtwarzania.

Narzędzia do debugowania programu

Produkt IBM MQ for z/OS udostępnia narzędzie śledzenia, którego można użyć do debugowania programów we wszystkich środowiskach.

Ponadto w środowisku CICS można używać następujących elementów:

- Program CICS Execution Diagnostic Facility (CEDF)
- Transakcja sterowania śledzeniem CICS (CETR)
- Wyjście funkcji API produktu 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 niezbędne w środowisku przetwarzania transakcyjnego, dzięki czemu przetwarzanie transakcji może być bezpiecznie używane.

Opcja ta jest w pełni obsługiwana przez produkt IBM MQ for z/OS w środowiskach CICS i IMS . Pełne wsparcie oznacza współpracę między menedżerami zasobów, dzięki czemu jednostki pracy mogą być zatwierdzane lub wycofane w unison, pod kontrolą systemów CICS lub IMS. Przykładami menedżerów zasobów są: Db2, CICS -Kontrola plików, IMSi IBM MQ for z/OS.

Aplikacje wsadowe produktu z/OS mogą używać wywołań programu IBM MQ for z/OS w celu nadania funkcji zatwierdzania jednofazowego. Oznacza to, że można zatwierdzić lub utworzyć kopię zapasową zdefiniowanego przez aplikację zestawu operacji kolejki, bez odwołania się do innych menedżerów zasobów.

Zatwierdzanie dwufazowe jest również obsługiwane w środowisku wsadowym produktu z/OS przy użyciu zarządzania transakcjami i odtwarzalnych usług menedżera zasobów (RRS). Więcej informacji na ten temat zawiera sekcja Punkty Syncpoints w aplikacjach wsadowych produktu z/OS.

Obsługa odtwarzania

Jeśli połączenie między menedżerem kolejek a systemem CICS lub IMS zostanie przerwane podczas transakcji, niektóre jednostki pracy mogą nie zostać pomyślnie wycofane.

Te jednostki pracy są jednak rozstrzygane przez menedżer kolejek (pod kontrolą menedżera punktu synchronizacji), gdy jego połączenie z systemem CICS lub IMS zostanie ponownie nawiązane.

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 środowiska, którego obsługuje.

Adaptery te są interfejsem między programami aplikacji a podsystemami IBM MQ for z/OS . Umożliwiają one programom korzystanie z interfejsu MQI.

Adapter wsadowy

Ta sekcja zawiera informacje na temat adaptera zadań wsadowych i protokołu zatwierdzania, które obsługuje.

Adapter wsadowy zapewnia dostęp do zasobów produktu IBM MQ for z/OS dla programów działających w:

- Tryb zadania (TCB)
- Stan problemu lub nadzorcy

- Podstawowy tryb sterowania przestrzenią adresową

Programy nie mogą być w trybie pamięci krzyżowej.

Połączenia między aplikacjami i programem IBM MQ for z/OS są dostępne na poziomie zadania. Adapter udostępnia pojedynczy wątek połączenia z bloku kontrolnego zadania aplikacji (TCB) do produktu IBM MQ for z/OS.

Adapter obsługuje protokół zatwierdzania jednofazowego w przypadku zmian dokonanych w zasobach należących do produktu IBM MQ for z/OS ; nie obsługuje protokołów multiphase-commit.

Adapter zadania wsadowego RRS
Ta sekcja zawiera informacje na temat adaptera wsadowego RRS oraz dwóch adapterów wsadowych RRS udostępnianych przez produkt IBM MQ.

Adapter usług zarządzania transakcjami i odtwarzalnymi usługami menedżera zasobów (RRS):

- Używa RRS produktu z/OS do kontroli zatwierdzania.
- Obsługuje jednoczesne połączenia z wieloma podsystemami IBM MQ działajnymi w pojedynczej instancji z/OS z jednego zadania.
- Udostępnia z/OSskoordynowaną kontrolę transakcji (za pomocą usługi RRS produktu z/OS) dla zasobów odtwarzalnych, do których dostęp jest uzyskiwany za pośrednictwem menedżerów odtwarzaczy zgodnych z RRS produktu z/OS dla następujących elementów:
  - Aplikacje, które łączą się z serwerem IBM MQ przy użyciu adaptera wsadowego RRS.
  - Db2-procedury składowane wykonywane w przestrzeni adresowej procedur składowanych Db2, która jest zarządzana przez menedżer obciążenia (WLM) w systemie z/OS.
- Obsługuje możliwość przełączania wątku wsadowego IBM MQ między obiektami TCBs.

Produkt IBM MQ for z/OS udostępnia dwa adaptery wsadowe RRS:

CSQBRSTB
Ten adapter wymaga zmiany dowolnej instrukcji MQCMIT na SRRCMIT i dowolnej instrukcji MQBACK na SRRBACK w aplikacji IBM MQ . (Jeśli kod MQCMIT lub MQBACK został zakodowany w aplikacji połączonej z CSQBRSTB, zostanie wyświetlony komunikat MQRC_ENVIRONMENT_ERROR).

CSQBRRSI
Ten adapter umożliwia aplikacji produktu IBM MQ korzystanie z MQCMIT i MQBACK albo SRRCMIT i SRRBACK.

Uwaga: CSQBRSTB i CSQBRRSI są dostarczane z atrybutami wiązań AMODE (31) RMODE (ANY). Jeśli aplikacja ładuje kod pośredniczący poniżej linii 16 MB, najpierw należy ponownie utworzyć kod pośredniczący z trybem RMODE (24).

Migracja

Istnieje możliwość przeprowadzenia migracji istniejących aplikacji Batch/TSO IBM MQ w celu użycia koordynacji RRS z kilkoma zmianami lub bez zmian.

Jeśli aplikacja IBM MQ zostanie ponownie dowiązana za pomocą adaptera CSQBRRSI, MQCMIT i MQBACK zostaną zsynchronizowane z jednostką pracy w produkcie IBM MQ i wszystkimi innymi menedżerami zasobów z włączoną obsługą protokołu RRS. W przypadku połączenia-edycja aplikacji IBM MQ z adapterem CSQBRSTB, należy zmienić MQCMIT na SRRCMIT i MQBACK na SRRBACK. To ostatnie podejście jest preferowane; wyraźnie wskazuje, że punkt synchronizacji nie jest ograniczony tylko do zasobów IBM MQ .

Adapter IMS
Jeśli adapter IMS jest używany z systemu IBM MQ for z/OS , należy upewnić się, że produkt IMS może uzyskać wystarczającą ilość pamięci, aby pomieścić komunikaty o długości do 100 MB.

Uwaga dla użytkowników

Adapter *IMS Adapter* zapewnia dostęp do zasobów produktu IBM MQ for z/OS dla następujących elementów:

- Programy przetwarzania komunikatów w trybie z połączeniem (MPPs)
- Interaktywne programy szybkiej ścieżki (IFPs)
- Wsadowe programy przetwarzania komunikatów (Batch Message processing programs-BMP)

Aby można było korzystać z tych zasobów, programy muszą być uruchomione w trybie zadania (TCB) i w stanie problemu; nie mogą być w trybie międzypamięciowym ani w trybie dostępu do rejestru.

Adapter udostępnia wątek połączenia z bloku kontrolnego zadania aplikacji (TCB) do produktu IBM MQ. Adapter obsługuje protokół zatwierdzania dwufazowego w celu wprowadzenia zmian w zasobach, których właścicielem jest IBM MQ for z/OS, z IMS występującym jako koordynator punktu synchronizacji.

Adapter udostępnia także program monitora wyzwalacza, który może uruchamiać programy automatycznie, gdy zostaną spełnione określone warunki wyzwalaczy w kolejce. Więcej informacji na ten temat zawiera sekcja "Uruchamianie aplikacji produktu IBM MQ przy użyciu wyzwalaczy" na stronie 958.

Jeśli używane są programy wsadowe DL/I, należy postępować zgodnie z wytycznymi podanymi w tym temacie dla programów wsadowych z/OS .

Pisanie aplikacji z/OS UNIX System Services

Adapter wsadowy obsługuje połączenia menedżera kolejek z obszarów zadań wsadowych i adresowych TSO:

Jeśli weźmiemy pod uwagę przestrzeń adresową partii, adapter obsługuje połączenia z wielu obiektów TCB w obrębie tej przestrzeni adresowej w następujący sposób:

- Każdy obiekt TCB może łączyć się z wieloma menedżerami kolejek za pomocą wywołania MQCONN lub MQCONNX (ale baza TCB może w dowolnym momencie mieć tylko jedną instancję połączenia z określonym menedżerem kolejek).
- Wiele rekordów wykonania instrukcji testowania może łączyć się z tym samym menedżerem kolejek (ale uchwyt menedżera kolejek zwrócony w dowolnym wywołaniu MQCONN lub MQCONNX jest powiązany z emitującym TCB i nie może być używany przez żaden inny obiekt TCB).

Program z/OS UNIX System Services obsługuje dwa typy wywołań pthread_create:

1. Wątki wagi ciężkiej, uruchamiaj jeden dla każdego TCB, które są ATTACHed i DETACHed na początku wątku i kończą się na z/OS.
2. Wątki o średniej wadze, uruchamiają jeden dla każdego TCB, ale TCB może być jedną z puli długotrwałych obiektów TCB. Aplikacja musi wykonać wszystkie niezbędne procedury czyszczące aplikacji, ponieważ jeśli jest połączona z serwerem, domyślne zakończenie wątku, które może zostać udostępnione przez serwer podczas kończenia zadania (TCB), **nie** jest zawsze sterowane.

Wątki lekkie nie są obsługiwane. (Jeśli aplikacja tworzy trwałe wątki, które wysyłają własne żądania pracy, ***aplikacja*** jest odpowiedzialna za czyszczenie zasobów przed uruchomieniem następnego zlecenia pracy).

IBM MQ for z/OS supports z/OS UNIX System Services threads using the Batch Adapter as follows:

1. Wątki heavyweight są w pełni obsługiwane jako połączenia wsadowe. Każdy wątek działa w swoim własnym TCB, który jest przyłączony i odłączany od początku i końca wątku. Jeśli wątek zostanie zakończony przed wywołaniem wywołania MQDISC, program IBM MQ for z/OS wykona standardową procedurę czyszczącą zadania, która obejmuje zatwierdzanie wszystkich zaległych jednostek pracy w przypadku, gdy wątek został zakończony normalnie, lub wycofuje się, jeśli wątek został zakończony nieprawidłowo.
2. Wątki o średniej wadze są w pełni obsługiwane, ale jeśli baza TCB będzie ponownie wykorzystywana przez inny wątek, aplikacja musi zapewnić, że wywołanie MQDISC, poprzedzone albo MQCMIT, albo MQBACK, zostanie wysłane przed rozpoczęciem następnego wątku. Oznacza to, że jeśli aplikacja ustała procedurę obsługi przerwania programu, a następnie aplikacja zostaje zakończona, procedura

obsługi przerwania musi wywołać wywołania MQCMIT i MQDISC przed ponownym użyciem bazy TCB dla innego wątku.

Uwaga: Modele wielowątkowe **nie** obsługują dostępu do wspólnych zasobów IBM MQ z wielu wątków.

Wyjście funkcji API-przejście dla z/OS

Ten temat zawiera informacje na temat interfejsu programistycznego, które są zależne od produktu.

Wyjście jest punktem w kodzie dostarczonym przez produkt IBM, w którym można uruchomić własny kod. Produkt IBM MQ for z/OS udostępnia *wyjście funkcji API* , które można wykorzystać do przechwytywania wywołań interfejsu MQI, a także do monitorowania lub modyfikowania funkcji wywołań MQI. W tej sekcji opisano sposób korzystania z wyjścia funkcji API, a także opis przykładowego programu obsługi wyjścia dostarczanego z produktem IBM MQ for z/OS.

Ta sekcja ma zastosowanie tylko w przypadku użytkowników produktu CICS TS V3.1 i wcześniejszych. Użytkownicy produktu CICS TS V3.2 i nowszego powinni zapoznać się z sekcją CICS Integracja z produktem IBM MQ w dokumentacji produktu CICS .

Uwaga

Wyjście funkcji API jest wywoływane tylko przez adapter CICS produktu IBM MQ for z/OS. Program obsługi wyjścia działa w przestrzeni adresowej CICS .

Pisanie własnego programu obsługi wyjścia
Można użyć przykładowego programu obsługi wyjścia funkcji API (CSQCAPX), który jest dostarczany razem z programem IBM MQ for z/OS jako środowisko dla własnego programu.

Jest to opisane w sekcji “Przykładowy program obsługi wyjścia API, CSQCAPX” na stronie 989.

Podczas zapisywania programu obsługi wyjścia w celu znalezienia nazwy wywołania MQI wywołanego przez aplikację należy zapoznać się z polem *ExitCommand* struktury MQXP. Aby znaleźć liczbę parametrów w wywołaniu, należy sprawdzić pole *ExitParmCount* . Można użyć 16-bajtowego pola *ExitUserArea* , aby zapisać adres dowolnej dynamicznej pamięci masowej, która jest zachowana przez aplikację. To pole jest zachowywane w wywołaniach wyjścia i ma ten sam czas życia, co zadanie CICS .

Jeśli używany jest produkt CICS Transaction Server V3.2, należy napisać program obsługi wyjścia, aby był wątkowo bezpieczny, i zadeklarować program obsługi wyjścia jako wątkowo bezpieczny. Jeśli używane są wcześniejsze wersje produktu CICS , zalecane jest również zapisywanie i deklarowanie programów obsługi wyjścia jako wątkowo bezpiecznych, aby były gotowe do migracji do produktu CICS Transaction Server V3.2.

Program obsługi wyjścia może zablokować wykonywanie wywołania MQI przez zwrócenie wartości MQXCC_SUPPRESS_FUNCTION lub MQXCC_SKIP_FUNCTION w polu *ExitResponse* . Aby umożliwić wykonanie wywołania (a program obsługi wyjścia zostanie ponownie wywołany po zakończeniu wywołania), program obsługi wyjścia musi zwrócić wartość MQXCC_OK.

Po wywołaniu po wywołaniu MQI program obsługi wyjścia może sprawdzać i modyfikować kody zakończenia i przyczyny ustawione przez wywołanie.

Użycie notatek

Oto kilka uwag ogólnych, które należy wziąć pod uwagę podczas pisania programu obsługi wyjścia:

- Ze względu na wydajność, napisz swój program w języku asembler-language. Jeśli użytkownik zapisuje go w dowolnym innym języku obsługiwanym przez produkt IBM MQ for z/OS, należy podać własny plik definicji danych.
- Link-edytuj swój program jako AMODE (31) i RMODE (ANY).
- Aby zdefiniować blok parametru wyjścia dla programu, należy użyć makra asemblera języka asemblera, CMQXPA.
- Określ WSPÓŁBIEŻNOŚĆ (THREADSAFE) podczas definiowania programu obsługi wyjścia i wszystkich programów, które wywołują program obsługi wyjścia.

- Jeśli używana jest opcja zabezpieczania pamięci masowej serwera CICS Transaction Server for z/OS , program musi być uruchamiany w kluczu wykonywania programu CICS . Oznacza to, że należy określić parametr EXECKEY (CICS) podczas definiowania zarówno programu obsługi wyjścia, jak i wszystkich programów, do których przechodzi sterowanie. Więcej informacji na temat programów obsługi wyjścia produktu CICS oraz narzędzia ochrony pamięci masowej CICS zawiera publikacja *CICS Customization Guide*.
- Program może korzystać z wszystkich interfejsów API (na przykład IMS, Db2i CICS). że może być używany program obsługi wyjścia użytkownika powiązany z zadaniem CICS . Może również korzystać z dowolnych wywołań MQI z wyjątkiem MQCONN, MQCONNX i MQDISC. Jednak wszystkie wywołania MQI w programie obsługi wyjścia nie wywołują drugiego czasu programu obsługi wyjścia.
- Program może wydawać komendy EXEC CICS SYNCPOINT lub EXEC CICS SYNCPOINT ROLLBACK. Jednak te komendy zatwierdzają lub wycofują ***wszystkie*** aktualizacje wykonywane przez zadanie aż do momentu, w którym wyjście było używane, a więc ich użycie nie jest zalecane.
- Program musi zakończyć się, wydając komendę EXEC CICS RETURN. Nie może on przenosić sterowania za pomocą komendy XCTL.
- Wyjścia są zapisywane jako rozszerzenia do kodu IBM MQ for z/OS . Upewnij się, że wyjście nie zakłóca żadnych programów lub transakcji produktu IBM MQ for z/OS , które korzystają z interfejsu MQI. Są one zwykle oznaczane przedrostkiem CSQ lub CK.
- Jeśli CSQCAPX jest zdefiniowany jako CICS, system CICS próbuje załadować program obsługi wyjścia, gdy program CICS połączy się z programem IBM MQ for z/OS. Jeśli ta próba zakończy się pomyślnie, komunikat CSQC301I zostanie wysłany do panelu CKQC lub do konsoli systemowej. Jeśli ładowanie nie powiedzie się (na przykład, jeśli moduł ładujący nie istnieje w żadnej z bibliotek w konkatenacji DFHRPL), komunikat CSQC315 zostanie wysłany do panelu CKQC lub do konsoli systemowej.
- Ponieważ parametry w obszarze komunikacji są adresami, program obsługi wyjścia musi być zdefiniowany jako lokalny w systemie CICS (czyli nie jest to program zdalny).

Przykładowy program obsługi wyjścia API, CSQCAPX
Przykładowy program obsługi wyjścia jest dostarczany jako program w języku asemblerowym. Plik źródłowy (CSQCAPX) jest dostarczany w bibliotece **thlqual**.SCSQASMS (gdzie **thlqual** jest kwalifikatorem wysokiego poziomu używanym przez instalację). Ten plik źródłowy zawiera pseudocodę, która opisuje logikę programu.

Przykładowy program zawiera kod inicjowania oraz układ, którego można użyć podczas pisania własnych programów obsługi wyjścia.

W przykładzie pokazano, w jaki sposób:

- Skonfiguruj blok parametru wyjścia
- Adres bloczków parametrów wywołania i wyjścia
- Określ, dla których wywołanie MQI wywołuje wyjście
- Określ, czy wyjście jest wywoływane przed lub po przetworzeniu wywołania MQI
- Umieszczenie komunikatu w tymczasowej kolejce pamięci masowej CICS
- Użyj makra DFHEIENT do dynamicznego akwizycji pamięci masowej, aby utrzymać ponowną wejściową
- Użyj funkcji DFHEIBLK dla bloku kontrolnego interfejsu exec CICS
- Warunki błędu pułapki
- Zwróć element sterujący do programu wywołującego

Projekt przykładowego programu obsługi wyjścia

Przykładowy program obsługi wyjścia zapisuje komunikaty do tymczasowej kolejki pamięci masowej CICS (CSQ1EXIT) w celu wyświetlenia operacji wyjścia.

Komunikaty wskazują, czy wyjście jest wywoływane przed wywołaniem MQI czy po wywołaniu interfejsu MQI. Jeśli wyjście zostanie wywołane po wywołaniu, komunikat zawiera kod zakończenia i kod przyczyny

zwrócony przez wywołanie. W przykładzie użyto stałych nazwanych z makra CMQXPA w celu sprawdzenia typu pozycji (czyli przed wywołaniem lub po wywołaniu).

Przykład nie wykonuje żadnej funkcji monitorowania, ale po prostu umieszcza w kolejce CICS stemplowane komunikaty informujące o typie wywołania, którego przetwarzanie jest przetwarzane. W ten sposób można określić wydajność interfejsu MQI, a także poprawne działanie programu obsługi wyjścia.

Uwaga: Przykładowy program obsługi wyjścia wysyła sześć wywołań EXEC CICS dla każdego wywołania MQI, które jest wykonywane w czasie działania programu. Jeśli używany jest ten program obsługi wyjścia, wydajność programu IBM MQ for z/OS jest obniżona.

Przygotowywanie i korzystanie z wyjścia funkcji API
Przykładowe wyjście jest dostarczane tylko w formularzu źródłowym.

Aby użyć przykładowego programu obsługi wyjścia lub programu obsługi wyjścia, który został napisany, należy utworzyć bibliotekę ładowania, tak jak w przypadku dowolnego innego programu CICS , zgodnie z opisem w sekcji "Budowanie aplikacji CICS w produkcie z/OS" na stronie 1133.

- For CICS Transaction Server for z/OS and CICS for MVS/ESA, when you update the CICS system definition (CSD) data set, the definitions you need are in the member **thlqual**.SCSQPROC(CSQ4B100).

  Uwaga: W definicjach używany jest przyrostek MQ. Jeśli ten przyrostek jest już używany w przedsiębiorstwie, musi on zostać zmieniony przed etapem składania.

Jeśli używane są domyślne definicje programu CICS , program obsługi wyjścia CSQCAPX jest instalowany w stanie ***wyłączony*** . Jest to spowodowane tym, że korzystanie z programu obsługi wyjścia może spowodować znaczne zmniejszenie wydajności.

Aby tymczasowo aktywować wyjście funkcji API, wykonaj następujące czynności:

1. Wydaj komendę **CEMT S PROGRAM(CSQCAPX) ENABLED** z głównego terminala CICS .
2. Uruchom transakcję CKQC i użyj opcji 3 w oknie rozwijanym Połączenie, aby zmienić status wyjścia funkcji API na **Włączone**.

If you want to run IBM MQ for z/OS with the API-crossing exit permanently enabled, with CICS Transaction Server for z/OS and CICS for MVS/ESA, do one of the following:

- Zmień definicję CSQCAPX w elemencie CSQ4B100, zmieniając STATUS (DISABLED) na STATUS (ENABLED). Definicję produktu CICS CSD można zaktualizować za pomocą dostarczonego przez CICSprogramu wsadowego DFHCSDUP.
- Zmień definicję CSQCAPX w grupie CSQCAT1 , zmieniając status z DISABLED na ENABLED.

W obu przypadkach należy ponownie zainstalować grupę. Można to zrobić za pomocą zimnego uruchamiania systemu CICS lub za pomocą transakcji CEDA CICS w celu reinstalacji grupy, gdy produkt CICS jest uruchomiony.

Uwaga: Użycie CEDA może spowodować wystąpienie błędu, jeśli którekolwiek z wpisów w grupie jest aktualnie używane.

Koniec informacji o interfejsie programistycznym wrażliwym na produkt.

Programowanie aplikacji z kolejkami współużytkowanymi

Ten temat zawiera informacje na temat niektórych czynników, które należy uwzględnić podczas projektowania nowych aplikacji w celu użycia współużytkowanych kolejek oraz podczas migrowania istniejących aplikacji do środowiska kolejki współużytkowanej.

Serializowanie aplikacji
Niektóre typy aplikacji mogą mieć pewność, że komunikaty są pobierane z kolejki dokładnie w takiej samej kolejności, w jakiej znajdują się w kolejce.

Na przykład, jeśli produkt IBM MQ jest używany do śledzenia aktualizacji bazy danych w systemie zdalnym, komunikat opisujący aktualizację rekordu musi zostać przetworzony po komunikacie opisującym wstawianie tego rekordu. W lokalnym środowisku kolejkowania jest to często realizowane przez aplikację

pobierającą komunikaty z kolejki za pomocą opcji MQOO_INPUT_EXCLUSIVE, uniemożliwiając w ten sposób inne pobieranie aplikacji z kolejki w tym samym czasie.

Program IBM MQ umożliwia aplikacjom otwieranie współużytkowanych kolejek 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 otrzymywać komunikaty dla aktualizacji tabeli A i aktualizacji tabeli B współbieżnie, to prosty mechanizm otwierania kolejki nie jest możliwy.

Jeśli ten typ aplikacji korzysta z wysokiej dostępności współużytkowanych kolejek, użytkownik może zdecydować, że inna instancja aplikacji, która uzyskuje dostęp do tych samych współużytkowanych kolejek, działająca w dodatkowym menedżerze kolejek, powinna przejąć, jeśli podstawowa aplikacja pobierający lub menedżer kolejek nie powiedzie się.

Jeśli podstawowy menedżer kolejek nie powiedzie się, wystąpią dwie rzeczy:

- Odtwarzanie równorzędne kolejki współużytkowanej zapewnia, że wszystkie niekompletne aktualizacje z aplikacji podstawowej zostaną zakończone lub wycofane.
- Dodatkowa aplikacja przejmuje przetwarzanie kolejki.

Dodatkowa aplikacja może zostać uruchomiona przed rozdaniem wszystkich niekompletnych jednostek pracy, co może spowodować, że aplikacja dodatkowa pobierze komunikaty z sekwencji. Aby rozwiązać ten typ problemu, aplikacja może wybrać *aplikację do serializacji*.

Aplikacja przekształcona do postaci szeregowej korzysta z wywołania MQCONNX w celu nawiązania połączenia z menedżerem kolejek, określając znacznik połączenia, gdy łączy się on z unikalną aplikacją. Wszystkie jednostki pracy wykonywane przez aplikację są oznaczone znacznikiem połączenia. Produkt IBM MQ zapewnia, że jednostki pracy w grupie współużytkowania kolejek z tym samym znacznikiem połączenia są przekształcane do postaci szeregowej (zgodnie z opcjami serializacji w wywołaniu MQCONNX).

Oznacza to, że jeśli aplikacja podstawowa używa wywołania MQCONNX ze znacznikiem połączenia `Database shadow retriever`, a dodatkowa aplikacja przejęcia podejmie próbę użycia wywołania MQCONNX z identycznym znacznikiem połączenia, dodatkowa aplikacja nie może połączyć się z drugim IBM MQ , dopóki nie zostaną zakończone żadne oczekujące podstawowe jednostki pracy, w tym przypadku przez odtwarzanie równorzędne.

Należy rozważyć użycie techniki aplikacji serializowanej dla aplikacji, które zależą od dokładnej sekwencji komunikatów w kolejce. W szczególności:

- Aplikacje, które nie mogą restartować się po awarii aplikacji lub menedżera kolejek do momentu zakończenia wszystkich operacji 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 inna instancja tej samej aplikacji jest już uruchomiona.

  W tym przypadku technika aplikacji przekształconej do postaci szeregowej jest wymagana tylko wtedy, gdy aplikacja nie może otworzyć kolejki na potrzeby danych wejściowych na wyłączność.

Uwaga: IBM MQ gwarantuje zachowanie sekwencji komunikatów, gdy spełnione są określone kryteria. Są one opisane w opisie komendy MQGET.

Aplikacje, które nie nadają się do użycia z kolejkami współużytkowanymi
Niektóre funkcje produktu IBM MQ nie są obsługiwane, gdy używane są kolejki współużytkowane, dlatego aplikacje korzystające z tych funkcji nie są odpowiednie dla środowiska kolejki współużytkowanej.

Podczas projektowania aplikacji kolejek współużytkowanych należy wziąć pod uwagę następujące kwestie:

- Indeksowanie kolejek jest ograniczone do współużytkowanych kolejek. Aby użyć identyfikatora komunikatu lub identyfikatora korelacji w celu wybrania komunikatu, który ma zostać zwrócony z kolejki, kolejka powinna zostać poindeksowana z poprawną wartością. Jeśli komunikaty są wybierane samodzielnie przez identyfikator komunikatu, kolejka wymaga typu indeksu MQIT_MSG_ID (chociaż

można również użyć komendy MQIT_NONE). Jeśli komunikaty są wybierane samodzielnie przez identyfikator korelacji, kolejka musi mieć typ indeksu MQIT_CORREL_ID.

- Nie można używać tymczasowych kolejek dynamicznych jako kolejek współużytkowanych. Można jednak korzystać z trwałych kolejek dynamicznych. Modele współużytkowanych kolejek dynamicznych mają wartość DEFTYPE SHAREDYN (współużytkowane dynamiczne), chociaż są one tworzone i niszczone w taki sam sposób, jak kolejki PERMDYN (trwałe dynamiczne).

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 135 na stronie 993.

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.

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 135 na stronie 993 opisuje, co należy wziąć pod uwagę przy podejmowaniu decyzji o typie kolejki inicjującej, która ma być używana.

Tabela 135. 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. 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 135. 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. **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. W przypadku wyzwalacza typu FIRST lub DEPTH jeden komunikat wyzwalacza jest generowany przez każdy menedżer kolejek, który ma otwartą nazwaną kolejkę inicjującą dla wejścia. **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 pomostowe IMS i IMS w systemie IBM MQ for z/OS

Te informacje ułatwiają pisanie aplikacji produktu IMS przy użyciu produktu IBM MQ.

- Informacje na temat używania punktów synchronizacji i wywołań MQI w aplikacjach IMS zawiera sekcja “Tworzenie aplikacji produktu IMS przy użyciu produktu IBM MQ” na stronie 71.
- Aby napisać aplikacje, które korzystają z mostu IBM MQ - IMS , należy zapoznać się z “Pisanie aplikacji mostu IMS” na stronie 75.

Użyj poniższych odsyłaczy, aby dowiedzieć się więcej na temat aplikacji mostu IMS i IMS w systemie IBM MQ for z/OS:

- “Tworzenie aplikacji produktu IMS przy użyciu produktu IBM MQ” na stronie 71
- “Pisanie aplikacji mostu IMS” na stronie 75

Pojęcia pokrewne

“Interfejs kolejki komunikatów-przegląd” na stronie 811
Informacje na temat komponentów interfejsu kolejek komunikatów (Message Queue Interface-MQI).

“Nawiąże połączenie z menedżerem kolejek i odłączenie go od niego” na stronie 824
Aby można było korzystać z usług programistycznych produktu IBM MQ , program musi mieć połączenie z menedżerem kolejek. Informacje zawarte w tej sekcji umożliwiają zapoznanie się z informacjami na temat nawiązywania połączenia z menedżerem kolejek i z jego rozłączeniem.

“Otwieranie i zamykanie obiektów” na stronie 833
Te informacje udostępniają wgląd w otwieranie i zamykanie obiektów produktu IBM MQ .

“Umieszczanie komunikatów w kolejce” na stronie 844
Te informacje umożliwiają zapoznanie się z informacjami na temat umieszczania komunikatów w kolejce.

“Pobieranie komunikatów z kolejki” na stronie 860
Ta sekcja zawiera informacje na temat pobierania komunikatów z kolejki.

“Sprawdzanie i ustawianie atrybutów obiektu” na stronie 943
Atrybuty to właściwości, które definiują parametry obiektu IBM MQ .

“Zatwierdzanie i wycofywanie jednostek pracy” na stronie 946
W tej sekcji opisano sposób zatwierdzania i tworzenia kopii zapasowych wszystkich możliwych do odtworzenia operacji get i put, które wystąpiły w jednostce pracy.

“Uruchamianie aplikacji produktu IBM MQ przy użyciu wyzwalaczy” na stronie 958
Informacje na temat wyzwalaczy i sposobów uruchamiania aplikacji IBM MQ przy użyciu wyzwalaczy.

“Praca z interfejsem MQI i klastrami” na stronie 978
Istnieją specjalne opcje dotyczące wywołań i kodów powrotu, które odnoszą się do technologii klastrowej.

“Używanie i zapisywanie aplikacji w systemie IBM MQ for z/OS” na stronie 983
Aplikacje produktu IBM MQ for z/OS mogą być uruchamiane z programów działających w wielu różnych środowiskach. Oznacza to, że mogą skorzystać z udogodnień dostępnych w więcej niż jednym środowisku.

Tworzenie aplikacji produktu IMS przy użyciu produktu IBM MQ

Podczas korzystania z produktu IBM MQ w aplikacjach IMS należy wziąć pod uwagę dodatkowe uwagi dotyczące tego, które wywołania funkcji API produktu MQ mogą być używane, a także mechanizm używany w punkcie synchronizacji.

Użyj poniższych odsyłaczy, aby dowiedzieć się więcej na temat pisania aplikacji IMS w systemie IBM MQ for z/OS:

- “Punkty synchronizacji w aplikacjach IMS” na stronie 71
- “Wywołania MQI w aplikacjach IMS” na stronie 72

Ograniczenia

Istnieją ograniczenia, których wywołania funkcji API produktu IBM MQ mogą być używane przez aplikację przy użyciu adaptera IMS .

Następujące wywołania funkcji API produktu IBM MQ nie są obsługiwane w ramach aplikacji przy użyciu adaptera IMS :

- MQCB
- MQCB_FUNCTION
- Komenda MQCTL

Pojęcia pokrewne

“Pisanie aplikacji mostu IMS” na stronie 75
Ten temat zawiera informacje na temat pisania aplikacji do korzystania z mostu IBM MQ - IMS .

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.

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 996
- “Aplikacje do uzyskiwania informacji” na stronie 999

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
```

Pisanie aplikacji mostu IMS

Ten temat zawiera informacje na temat pisania aplikacji do korzystania z mostu IBM MQ - IMS .

Informacje na temat mostu IBM MQ - IMS znajdują się w sekcji Most IMS.

Użyj poniższych odsyłaczy, aby dowiedzieć się więcej na temat pisania aplikacji mostu IMS w systemie IBM MQ for z/OS:

- “Sposób, w jaki most IMS obsługuje komunikaty” na stronie 75
- “Pisanie programów transakcyjnych IMS za pomocą programu IBM MQ” na stronie 1006

Pojęcia pokrewne

“Tworzenie aplikacji produktu IMS przy użyciu produktu IBM MQ” na stronie 71
Podczas korzystania z produktu IBM MQ w aplikacjach IMS należy wziąć pod uwagę dodatkowe uwagi dotyczące tego, które wywołania funkcji API produktu MQ mogą być używane, a także mechanizm używany w punkcie synchronizacji.

Sposób, w jaki most IMS obsługuje komunikaty

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 produktu IBM MQ , które zostały zdefiniowane z klasą pamięci masowej, która określa grupę XCF i nazwę elementu docelowego systemu IMS . Są one nazywane kolejkami mostów MQ-IMS lub po prostu kolejkami **bridge** .

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

Przed wysłaniem komunikatów do aplikacji IMS użytkownik nie musi logować się do produktu IMS . Identyfikator użytkownika w polu *UserIdentifier* struktury MQMD jest używany do sprawdzania zabezpieczeń. Poziom sprawdzania jest określany, gdy program IBM MQ łączy się z serwerem IMSi jest opisany w sekcji Kontrola dostępu do aplikacji dla mostu IMS. Umożliwia to zaimplementowanie pseudo signon.

Most IBM MQ - IMS akceptuje następujące typy komunikatów:

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

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

  Uwaga:

  1. Nawiasy kwadratowe, [], reprezentują opcjonalne wielosegmenty.
  2. Ustaw pole *Format* struktury MQMD na wartość MQFMT_IMS w celu użycia 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 plus długość wartości MQIIH (jeśli jest obecna) jest równa długości komunikatu.

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

- Jeśli komunikat zawiera strukturę MQIIH, most weryfikuje wartość 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 za pomocą komunikatu DFS1288E . Jeśli kod transakcji reprezentuje komendę, program IMS wykonuje komendę; w przeciwnym razie komunikat znajduje się w kolejce IMS dla transakcji.
- Jeśli komunikat zawiera dane transakcji IMS , ale nie ma struktury MQIIH, most IMS tworzy 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 (zatwierdź-następnie-wyślij)
  - *Format* w strukturze MQMD jest używany jako *MFSMapName* (na wejściu)
  - Tryb zabezpieczeń to MQISS_CHECK

  Komunikat odpowiedzi jest również zbudowany bez struktury MQIIH, przyjmując *Format* dla deskryptora MQMD z *MFSMapName* danych wyjściowych IMS .

Most IBM MQ - IMS korzysta z jednej lub dwóch potoków dla każdej kolejki IBM MQ :

- Zsynchronizowana rura jest używana dla wszystkich komunikatów przy użyciu trybu zatwierdzania 0 (COMMIT_THEN_SEND) (te pokazy ze SYN w polu statusu komendy IMS /DIS TMEMBER klienta TPIPE xxxx)
- Niezsynchronizowana Tpipe jest używana dla wszystkich komunikatów przy użyciu trybu kontroli transakcji 1 (SEND_THEN_COMMIT).

Potoki są tworzone przez produkt IBM MQ , gdy są one używane jako pierwsze. Niezsynchronizowana potok Tpipe istnieje do momentu zrestartowania serwera IMS . Zsynchronizowane trury istnieją do momentu, gdy program IMS nie jest zimny. Nie można samodzielnie usunąć tych potoków.

Więcej informacji na temat sposobu obsługi mostu IBM MQ - IMS z komunikatami można znaleźć w następujących tematach:

- "Odwzorowanie komunikatów IBM MQ na typy transakcji IMS" na stronie 77
- "Jeśli komunikat nie może zostać umieszczony w kolejce produktu IMS" na stronie 77
- "Kody sprzężenia zwrotnego mostu IMS" na stronie 78
- "Pola MQMD w komunikatach z mostu IMS" na stronie 78
- "Pola MQIIH w komunikatach z mostu IMS" na stronie 79
- "Komunikaty odpowiedzi z programu IMS" na stronie 80
- "Korzystanie z alternatywnych PCB odpowiedzi w transakcjach IMS" na stronie 81
- "Wysyłanie niezamówionych komunikatów z produktu IMS" na stronie 81
- "Segmentacja komunikatów" na stronie 81
- "Konwersja danych dla komunikatów do i z mostu IMS" na stronie 81

Pojęcia pokrewne

"Pisanie programów transakcyjnych IMS za pomocą programu IBM MQ" na stronie 1006
Kodowanie wymagane do obsługi transakcji IMS za pomocą produktu IBM MQ zależy od formatu komunikatu wymaganego przez transakcję IMS oraz zakresu odpowiedzi, które może zwrócić. Jednak w sytuacji, gdy aplikacja obsługuje informacje o formatowaniu ekranu produktu IMS , należy wziąć pod uwagę kilka punktów.

Odwzorowanie komunikatów IBM MQ na typy transakcji IMS

Tabela opisująca odwzorowanie komunikatów IBM MQ na typy transakcji IMS .

Tabela 136. 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 • Transakcje nienaprawialne są odrzucane przez IMS	• Transakcje krótkiej ścieżki • Transakcje konwersacyjne • W pełni funkcjonalne transakcje
Nietrwałe komunikaty IBM MQ	• Nienaprawialne transakcje z pełnymi funkcjami • Transakcje odtwarzalne są dozwolone w poprawkach IMS V8 i APAR PQ61404 oraz we wszystkich późniejszych wersjach systemu IMS .	• Transakcje krótkiej ścieżki • Transakcje konwersacyjne • 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) .

Jeśli komunikat nie może zostać umieszczony w kolejce produktu IMS

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

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

- Jeśli komunikat nie może zostać umieszczony w programie IMS , ponieważ komunikat jest niepoprawny, komunikat jest umieszczany w kolejce niedostarczonych komunikatów, a do konsoli systemowej wysyłany jest komunikat.
- 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 komunikat IBM MQ zawierający przyczynę niepowodzenia w kolejce odpowiedzi.

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

  Aby ponownie wysłać komunikaty, należy wykonać ***jeden*** z następujących działań:

  – Zatrzymaj i zrestartuj potoki w programie IMS , które odpowiadają kolejce
  – Zmień kolejkę na GET (WYŁĄCZONE), a następnie ponownie w celu uzyskania (ENABLED)
  – Zatrzymaj i zrestartuj produkt IMS lub OTMA
  – Zatrzymaj i zrestartuj podsystem IBM MQ .
- Jeśli komunikat został odrzucony przez produkt IMS w przypadku innych niż błąd komunikatów, komunikat IBM MQ zostanie zwrócony do kolejki źródłowej, program IBM MQ zatrzyma przetwarzanie kolejki, a do konsoli systemowej zostanie wysłany komunikat o błędzie.

  Jeśli wymagany jest komunikat z raportem o wyjątku, most umieszcza go w kolejce odpowiedzi z uprawnieniami inicjatora. Jeśli komunikat nie może zostać umieszczony w kolejce, komunikat raportu jest umieszczany w kolejce niedostarczonych komunikatów z uprawnieniami mostu. Jeśli nie można go umieścić w kolejce DLQ, zostanie ona odrzucona.

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.

Pola MQMD w komunikatach z mostu IMS
Informacje na temat pól MQMD w komunikatach z mostu IMS .

Kod MQMD komunikatu źródłowego jest przenoszony przez produkt IMS w sekcji Dane użytkownika w nagłówkach OTMA. Jeśli komunikat pochodzi z produktu IMS, jest on budowany przy użyciu wyjścia rozstrzygania miejsca docelowego produktu IMS . Kod MQMD komunikatu odebranego z produktu IMS jest zbudowany w następujący sposób:

StrucID
"MD"

Wersja
MQMD_VERSION_1

Raport
MQRO_NONE

MsgType
MQMT_REPLY

Utrata ważności
Jeśli wartość MQIIH_PASS_EXPIRATION jest ustawiona w polu flagi w tabeli MQIIH, to pole zawiera pozostały czas utraty ważności, w przeciwnym razie jest on ustawiony na wartość MQEI_UNLIMITED.

Opinie
MQFB_NONE

Kodowanie
MQENC.Native (kodowanie systemu z/OS)

CodedCharSetId
MQCCSI_Q_MGR (CodedCharSetID w systemie 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 kontroli transakcji: 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 (ustawiane na lokalną nazwę 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 ma błędu, w przeciwnym razie nazwa QMGR

PutDate
Data umieszczenia komunikatu

PutTime
Czas umieszczenia komunikatu

Dane_pochodzenia_aplikacji
Puste

Pola MQIIH w komunikatach z mostu IMS
Informacje na temat pól MQIIH w komunikatach z mostu IMS .

Obiekt MQIIH komunikatu odebranego z produktu IMS jest zbudowany w następujący sposób:

StrucId
"IIH"

Wersja
1

StrucLength
84

Kodowanie
MQENC_NATIVE

CodedCharSetId
MQCCSI_Q_MGR

Format
MQIIH.ReplyToFormat komunikatu wejściowego, jeśli MQIIH.ReplyToFormat nie jest pusta, w przeciwnym razie IOPCB.MODNAME

Flagi
0

LTermOverride
Nazwa LTERM (Tpipe) z nagłówka OTMA

MFSMapName
Nazwa odwzorowania z nagłówka OTMA

Format ReplyTo
Puste

Authenticator
MQIIH.Authenticator komunikatu wejściowego, jeśli komunikat odpowiedzi jest umieszczany w kolejce mostu MQ-IMS . W przeciwnym razie puste pole jest puste.

Identyfikator TranInstance
Identyfikator konwersacji/znacznik serwera z nagłówka OTMA, jeśli konwersacja jest konwersacja. W wersjach systemu IMS wcześniejszych niż V14, to pole zawsze jest puste, jeśli nie jest w konwersacji. Począwszy od systemu IMS V14 , pole to może być ustawione przez system IMS , nawet jeśli nie 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

Komunikaty odpowiedzi z programu IMS
Gdy transakcja IMS jest transakcją ISRTs z jej IOPCB, komunikat jest kierowany z powrotem do źródłowego LTERM lub TPIPE.

Są one widoczne w produkcie IBM MQ jako komunikaty odpowiedzi. Komunikaty odpowiedzi z programu IMS są umieszczane w kolejce odpowiedzi określonej w pierwotnym komunikacie. Jeśli komunikat nie może zostać umieszczony 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 programu IMS zostanie wysłane potwierdzenie negatywne, co oznacza, że komunikat nie może zostać odebrany. Odpowiedzialność za komunikat jest następnie zwracana do programu IMS. Jeśli używany jest tryb kontroli transakcji 0, komunikaty z tego protokołu Tpipe nie są wysyłane do mostu i pozostają w kolejce produktu IMS . To znaczy, że kolejne komunikaty nie są wysyłane do czasu restartu. Jeśli używany jest tryb kontroli transakcji 1, inne prace mogą być kontynuowane.

Jeśli odpowiedź ma strukturę MQIIH, jej typem formatu jest MQFMT_IMS;, jeśli nie, jego typ formatu jest określany przez nazwę MOD produktu IMS używaną podczas wstawiania komunikatu.

Korzystanie z alternatywnych PCB odpowiedzi w transakcjach IMS
Gdy transakcja IMS korzysta z alternatywnych PCB odpowiedzi (ISRTs do ALTPCB lub wysyła wywołanie CHNG do modyfikowalnego bloku PCB), wywołanie wyjścia routingu wstępnego (DFSYPRX0) jest wywoływane w celu określenia, czy komunikat powinien zostać ponownie uruchomiony.

Jeśli komunikat ma zostać przekierowany, następuje wywołanie wyjścia rozstrzygania miejsca docelowego (DFSYDRU0) w celu potwierdzenia miejsca docelowego i przygotowania informacji o nagłówku. Patrz sekcja Korzystanie z wyjść OTMA w programie IMS i Wyjście z routingu wstępnego DFSYPRX0 , aby uzyskać informacje na temat tych programów obsługi wyjścia.

Jeśli działanie nie zostanie wykonane w wyjściach, wszystkie dane wyjściowe z IMS transakcji zainicjowanych z menedżera kolejek produktu IBM MQ , niezależnie od tego, czy mają być wykonywane przez IOPCB, czy też do grupy ALTPCB, zostaną zwrócone do tego samego menedżera kolejek.

Wysyłanie niezamówionych komunikatów z produktu IMS
Aby wysyłać komunikaty z produktu IMS do kolejki produktu IBM MQ , należy wywołać transakcję IMS , która jest używana przez ISRTs do bazy danych ALTPCB.

Aby kierować niezamówionym komunikatami z produktu IMS i zbudować dane użytkownika OTMA, należy napisać procedury poprzedzające routing i rozwiązywanie miejsca docelowego, tak aby można było poprawnie zbudować deskryptor MQMD komunikatu. Informacje na temat tych programów obsługi wyjścia można znaleźć w sekcji Wyjście routingu wstępnego DFSYPRX0 i Wyjście użytkownika rozstrzygania miejsca docelowego .

Uwaga: Most IBM MQ - IMS nie wie, czy komunikat, który otrzymuje, jest odpowiedzią, czy niezamówionym komunikatem. Obsługuje on ten sam komunikat w każdym przypadku, budując tabelę MQMD i MQIIH odpowiedzi na podstawie UserData OTMA, które dotarło do komunikatu.

Niezamówione komunikaty mogą tworzyć nowe potoków. Na przykład, jeśli istniejąca transakcja IMS została przełączona na nowy LTERM (na przykład PRINT01), ale implementacja wymaga, aby dane wyjściowe były dostarczane za pośrednictwem OTMA, zostanie utworzona nowa Tpipe (w tym przykładzie o nazwie PRINT01). Domyślnie jest to Tpipe, który nie jest zsynchronizowany. Jeśli implementacja wymaga, aby komunikat mógł być odtwarzalny, należy ustawić flagę wyjścia wyjścia rozstrzygania miejsca docelowego. Więcej informacji na ten temat zawiera podręcznik *IMS Customization Guide* .

Segmentacja komunikatów
Transakcje IMS można zdefiniować jako oczekiwane dane wejściowe jedno-lub wielosegmentowe.

Źródłowa aplikacja IBM MQ musi konstruować dane wejściowe użytkownika po strukturze MQIIH jako co najmniej jeden segment danych LLZZ. Wszystkie segmenty komunikatu IMS muszą być zawarte w pojedynczym komunikacie IBM MQ wysłanym z pojedynczą MQPUT.

Maksymalna długość segmentu danych LLZZ jest definiowana przez system IMS/OTMA (32767 bajtów). Całkowita długość komunikatu IBM MQ jest sumą bajtów LL oraz długością struktury MQIIH.

Wszystkie segmenty odpowiedzi są zawarte w jednym komunikacie IBM MQ .

Istnieje dalsze ograniczenie dotyczące ograniczenia 32 kB dla komunikatów o formacie MQFMT_IMS_VAR_STRING. Gdy dane w komunikacie ASCII o mieszanym identyfikatorze CCSID są konwertowane na komunikat o kodowaniu EBCDIC mieszanym, bajt shift-in lub shift-out jest dodawany za każdym razem, gdy istnieje przejście między znakami SBCS i DBCS. Ograniczenie o wielkości 32 kB ma zastosowanie do maksymalnej wielkości komunikatu. Oznacza to, że ponieważ pole LL w komunikacie

nie może przekroczyć 32 kB, komunikat nie może być większy niż 32 kB, w tym wszystkie znaki shift-in i shift-out. Aplikacja budowania komunikatu musi zezwalać na to działanie.

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 zawierającej informacje XCF zdefiniowane dla swojej klasy pamięci masowej. Konwersja danych nie jest wykonywana, gdy komunikat jest dostarczany do kolejki publikowania/subskrybowania.

Wszystkie wymagane wyjścia muszą być dostępne dla rozproszonego narzędzia kolejkowania w zestawie danych przywoływanym przez instrukcję CSQXLIB DD. Oznacza to, że można wysyłać komunikaty do aplikacji IMS , korzystając z mostu IBM MQ - IMS z dowolnej platformy IBM MQ .

Jeśli wystąpiły błędy konwersji, komunikat jest umieszczany w kolejce bez konwersji. W rezultacie w końcu jest on traktowany jako błąd przez most IBM MQ - IMS , ponieważ most nie może rozpoznać formatu nagłówka. Jeśli wystąpi błąd konwersji, do konsoli z/OS zostanie wysłany komunikat o błędzie.

Szczegółowe informacje na temat konwersji danych można znaleźć w sekcji “Pisanie wyjść konwersji danych” na stronie 1082 .

Wysyłanie komunikatów do mostu IBM MQ - IMS

Aby upewnić się, że konwersja jest wykonywana poprawnie, należy poinformować menedżera kolejek o tym, jaki jest format komunikatu.

Jeśli komunikat ma strukturę MQIIH, wartość *Format* w strukturze MQMD musi być ustawiona na wbudowany format MQFMT_IMS, a wartość *Format* w tabeli MQIIH musi być ustawiona na nazwę formatu opisowego, który opisuje dane komunikatu. Jeśli nie ma tabeli MQIIH, ustaw wartość *Format* w strukturze MQMD na nazwę formatu.

Jeśli dane (inne niż LLZZs) to wszystkie dane znakowe (MQCHAR), należy użyć jako nazwy formatu (w tabeli MQIIH lub MQMD, w zależności od przypadku), wbudowanego formatu MQFMT_IMS_VAR_STRING. W przeciwnym razie należy użyć własnej nazwy formatu, w którym to przypadku należy również podać wyjście konwersji danych dla formatu. Wyjście musi obsługiwać konwersję LLZZs w swoim komunikacie, oprócz samych danych (ale nie musi obsługiwać żadnej MQIIH na początku wiadomości).

Jeśli aplikacja używa produktu *MFSMapName*, zamiast tego można użyć komunikatów z produktem MQFMT_IMS i zdefiniować nazwę odwzorowania przekazanej do transakcji IMS w polu MFSMapName tabeli MQIIH.

Odbieranie komunikatów z mostu IBM MQ - IMS

Jeśli w oryginalnym komunikacie, który jest wysyłany do programu IMS, znajduje się struktura MQIIH, jest ona również obecna w komunikacie odpowiedzi.

Aby upewnić się, że odpowiedź została poprawnie przekształcona:

- Jeśli w pierwotnym komunikacie znajduje się struktura MQIIH, w polu MQIIH *ReplytoFormat* oryginalnego komunikatu należy określić format, który ma być wyświetlany dla komunikatu odpowiedzi. Ta wartość jest umieszczana w polu MQIIH *Format* komunikatu odpowiedzi. Jest to szczególnie przydatne w przypadku, gdy wszystkie dane wyjściowe mają postać LLZZ < dane znakowe >.
- Jeśli w oryginalnym komunikacie nie ma struktury MQIIH, określ format, który ma być wyświetlany dla komunikatu odpowiedzi jako nazwa MFS MOD w ISRT aplikacji IMS do IOPCB.

Pisanie programów transakcyjnych IMS za pomocą programu IBM MQ
Kodowanie wymagane do obsługi transakcji IMS za pomocą produktu IBM MQ zależy od formatu komunikatu wymaganego przez transakcję IMS oraz zakresu odpowiedzi, które może zwrócić. Jednak w sytuacji, gdy aplikacja obsługuje informacje o formatowaniu ekranu produktu IMS , należy wziąć pod uwagę kilka punktów.

Gdy transakcja IMS jest uruchamiana z ekranu 3270, komunikat przechodzi przez program IMS Message Format Services. Może to usunąć całą zależność terminalową od strumienia danych obserwowanego 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 przypadku niektórych transakcji produktu IMS aplikacja użytkownika końcowego może zmodyfikować określone zachowanie ekranu 3270, na przykład podświetlając pole, w którym wprowadzono niepoprawne dane. Ten typ informacji jest powiadamiany przez dodanie dwubajtowego pola atrybutu do komunikatu IMS dla każdego pola ekranu, które musi zostać zmodyfikowane przez program.

Oznacza to, że w przypadku kodowania aplikacji w celu naśladowania terminalu 3270 należy uwzględnić te pola podczas budowania lub odbierania komunikatów.

Konieczne może być zakodowanie informacji w programie do przetwarzania:

- Naciśnięty klawisz (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
  - Natężenie wysokie, normalne lub zerowe
  - Kolor
  - Określa, czy program IMS oczekuje pola z powrotem przy następnym naciśnięciu klawisza Enter.
- To, czy w aplikacji IMS użyto znaków o kodzie zero (X'3F') w dowolnych polach.

Jeśli komunikat IMS zawiera tylko dane znakowe (poza segmentem LLZZ-data) i używana jest struktura MQIIH, ustaw format MQMD na MQFMT_IMS i format MQIIH na MQFMT_IMS_VAR_STRING.

Jeśli komunikat IMS zawiera tylko dane znakowe (oprócz segmentu danych LLZZ), a **nie** jest używany struktura MQIIH, należy ustawić format MQMD na wartość MQFMT_IMS_VAR_STRING i upewnić się, że aplikacja IMS określa parametr MODname MQFMT_IMS_VAR_STRING podczas odpowiadania na wartość. Jeśli wystąpi problem (na przykład użytkownik nieuprawniony do używania transakcji), a program IMS wyśle komunikat o błędzie, ma on nazwę MODname formularza DFSMOx, gdzie x jest liczbą z zakresu od 1 do 5. Ta opcja jest umieszczana w strukturze MQMD.Format.

Jeśli komunikat IMS zawiera dane binarne, zapakowane lub zmiennopozycyjne (poza segmentem danych LLZZ), należy zakodować własne procedury konwersji danych. Więcej informacji na temat formatowania ekranu programu IMS można znaleźć w podręczniku *IMS/ESA Application Programming: Transaction Manager* .

Podczas pisania kodu w celu obsługi transakcji produktu IMS za pomocą produktu IBM MQnależy rozważyć następujące tematy.

- “Zapisywanie aplikacji produktu IBM MQ w celu wywołania transakcji konwersacyjnych produktu IMS” na stronie 1007
- “Pisanie programów zawierających komendy IMS” na stronie 1008
- “Wyzwalanie” na stronie 1008

Zapisywanie aplikacji produktu IBM MQ w celu wywołania transakcji konwersacyjnych produktu IMS

Use this information as a guide for considerations when writing IBM MQ application to invoke IMS conversational transactions.

Podczas pisania aplikacji, która wywołuje konwersację IMS , należy wziąć pod uwagę następujące kwestie:

- Dołącz strukturę MQIIH do komunikatu aplikacji.
- Ustaw wartość parametru *CommitMode* w tabeli MQIIH na MQICM_SEND_THEN_COMMIT.

- Aby wywołać nową konwersację, należy ustawić wartość *TranState* w tabeli MQIIH na wartość MQITS_NOT_IN_CONVERSATION.
- Aby wywołać drugi i kolejne kroki konwersacji, należy ustawić wartość parametru *TranState* na MQITS_IN_CONVERSATION, a następnie ustawić wartość *TranInstanceId* na wartość pola zwróconego w poprzednim kroku konwersacji.
- W programie IMS nie ma prostego sposobu na znalezienie wartości *TranInstanceId*. Należy utracić oryginalny komunikat wysłany z produktu IMS.
- Aplikacja musi sprawdzić *TranState* komunikatów z programu IMS , aby sprawdzić, czy transakcja IMS zakończyła konwersację.
- Aby zakończyć konwersację, można użyć opcji /EXIT. Należy również zacytować *TranInstanceId*, ustawić wartość *TranState* na MQITS_IN_CONVERSATION, a następnie użyć kolejki IBM MQ , w której konwersacja jest wykonywana.
- Nie można używać /HOLD ani /REL do przechowywania lub zwalniania konwersacji.
- Konwersacje wywoływane przez most IBM MQ - IMS są przerywane, jeśli program IMS zostanie zrestartowany.

Pisanie programów zawierających komendy IMS

Program użytkowy może utworzyć komunikat IBM MQ z formularza LLZZ*komenda*zamiast transakcji, gdzie *komenda* ma postać /DIS TRAN PART lub /DIS POOL ALL.

Większość komend produktu IMS może zostać wydana w ten sposób. Szczegółowe informacje na ten temat zawiera sekcja *IMS V11 Communications and Connections* . Dane wyjściowe komendy są odbierane w komunikacie odpowiedzi IBM MQ w postaci tekstowej, która zostanie wysłana do terminalu 3270 w celu wyświetlenia.

Program OTMA zaimplementował specjalny formularz komendy wyświetlania transakcji IMS , który zwraca archiwowaną formę danych wyjściowych. Dokładny format jest zdefiniowany w publikacji *IMS V11 Communications and Connections*. Aby wywołać ten formularz z komunikatu programu IBM MQ , należy zbudować dane komunikatu tak jak wcześniej, na przykład /DIS TRAN PART, i ustawić pole TranState w tabeli MQIIH na wartość MQITS_ARCHITECTED. IMS przetwarza komendę i zwraca odpowiedź w postaci architected. Odpowiedź architected zawiera wszystkie informacje, które można znaleźć w postaci tekstowej danych wyjściowych oraz jeden dodatkowy fragment informacji: czy transakcja jest definiowana jako odtwarzalna czy nie do odtworzenia.

Wyzwalanie

Most IBM MQ - IMS nie obsługuje komunikatów wyzwalacza.

W przypadku zdefiniowania kolejki inicjuj, która korzysta z klasy pamięci masowej z parametrami XCF, komunikaty umieszczone w tej kolejce są odrzucane po dostaniu się do mostu.

Pisanie aplikacji proceduralnych klienta

Co należy wiedzieć, aby pisać aplikacje klienckie w systemie IBM MQ , korzystając z języka proceduralnego.

Aplikacje mogą być budowane i uruchamiane w środowisku klienta IBM MQ . Aplikacja musi być zbudowana i dowiązana do używanej partycji IBM MQ MQI client . Sposób budowania i tworzenia powiązanych aplikacji różni się w zależności od używanej platformy i języka programowania. Informacje na temat budowania aplikacji klienckich można znaleźć w sekcji “Budowanie aplikacji dla produktu IBM MQ MQI clients” na stronie 1015.

Aplikację IBM MQ można uruchomić zarówno w pełnym środowisku IBM MQ , jak i w środowisku IBM MQ MQI client bez zmiany kodu pod warunkiem, że spełnione są określone warunki. Więcej informacji na temat uruchamiania aplikacji w środowisku klienta IBM MQ zawiera sekcja “Uruchamianie aplikacji w środowisku IBM MQ MQI client” na stronie 1017.

Jeśli do zapisu aplikacji w środowisku IBM MQ MQI client jest używany interfejs kolejki komunikatów (MQI), to podczas wywołania MQI konieczne jest wprowadzenie dodatkowych elementów sterujących w celu upewniania się, że przetwarzanie aplikacji IBM MQ nie jest zakłócane. Więcej informacji na temat tych elementów sterujących zawiera sekcja “Korzystanie z interfejsu MQI w aplikacji klienckiej” na stronie 1009.

Poniższe tematy zawierają informacje na temat przygotowywania i uruchamiania innych typów aplikacji jako aplikacji klienckich:

- “Przygotowywanie i uruchamianie aplikacji CICS i Tuxedo” na stronie 1030
- “Przygotowywanie i uruchamianie aplikacji produktu Microsoft Transaction Server” na stronie 49
- “Przygotowywanie i uruchamianie aplikacji IBM MQ JMS” na stronie 1032

Pojęcia pokrewne

“Pojęcia związane z projektowaniem aplikacji” na stronie 7
Do pisania aplikacji IBM MQ można użyć wyboru języków proceduralnych lub obiektowych. Przed rozpoczęciem projektowania i pisania aplikacji produktu IBM MQ należy zapoznać się z podstawowymi pojęciami dotyczącymi produktu IBM MQ .

“Tworzenie aplikacji dla składnika IBM MQ” na stronie 5
Istnieje możliwość tworzenia aplikacji w celu wysyłania i odbierania komunikatów oraz do zarządzania menedżerami kolejek i powiązanymi zasobami. Produkt IBM MQ obsługuje aplikacje napisane w wielu różnych językach i w różnych ramach.

“Uwagi dotyczące projektowania aplikacji produktu IBM MQ” na stronie 50
Po zdecydowaniu, w jaki sposób aplikacje mogą korzystać z platform i środowisk, które są dostępne dla użytkownika, należy zdecydować, w jaki sposób korzystać z funkcji oferowanych przez produkt IBM MQ.

“Pisanie aplikacji proceduralnej w celu kolejkowania” na stronie 810
Ta sekcja zawiera informacje na temat pisania aplikacji kolejkowania, łączenia się i rozłączania z menedżerem kolejek, publikowania/subskrypcji oraz obiektów otwierających i zamykających.

“Zapisywanie aplikacji publikowania/subskrypcji” na stronie 901
Rozpocznij pisanie aplikacji publikowania/subskrypcji IBM MQ .

“Budowanie aplikacji proceduralnej” na stronie 1099
Aplikację IBM MQ można napisać w jednym z kilku języków proceduralnych, a następnie uruchomić aplikację na kilku różnych platformach.

“Obsługa proceduralnych błędów programu” na stronie 1145
Te informacje wyjaśniają błędy związane z wywołaniami MQI aplikacji, gdy wywołuje połączenie, lub gdy jego komunikat jest dostarczany do miejsca docelowego.

Zadania pokrewne

“Korzystanie z przykładowych programów proceduralnych produktu IBM MQ” na stronie 1165
Te przykładowe programy są zapisywane w językach proceduralnych i demonstrują typowe zastosowania interfejsu kolejki komunikatów (Message Queue Interface-MQI). Programy IBM MQ na różnych platformach.

Korzystanie z interfejsu MQI w aplikacji klienckiej

W tej kolekcji tematów są rozważane różnice między zapisaniem aplikacji IBM MQ w celu uruchomienia w środowisku klienta interfejsu kolejek komunikatów (MQI) i uruchamianie w pełnym środowisku menedżera kolejek produktu IBM MQ .

Podczas projektowania aplikacji należy wziąć pod uwagę, jakie elementy sterujące należy nałożyć podczas wywołania MQI, aby upewnić się, że przetwarzanie aplikacji IBM MQ nie zostało zakłócone.

Przed uruchomieniem aplikacji, które korzystają z interfejsu MQI, należy utworzyć określone obiekty produktu IBM MQ . Więcej informacji na ten temat zawiera sekcja Programy aplikacji korzystające z interfejsu MQI.

Ograniczanie wielkości komunikatu w aplikacji klienckiej

Menedżer kolejek ma maksymalną długość komunikatu, ale maksymalna wielkość komunikatu, jaki można przesłać z aplikacji klienckiej, jest ograniczona przez definicję kanału.

Atrybut maksymalnej długości komunikatu (MaxMsgLength) menedżera kolejek to maksymalna długość komunikatu, który może być obsługiwany przez tego menedżera kolejek.

Multi W systemie Wiele platformmożna zwiększyć atrybut maksymalnej długości komunikatu menedżera kolejek. Więcej informacji na ten temat zawiera sekcja ALTER QMGR.

Za pomocą wywołania MQINQ można znaleźć wartość parametru MaxMsg(Maksymalna długość komunikatu) dla menedżera kolejek.

Jeśli atrybut długości MaxMsgjest zmieniany, nie jest wykonywane sprawdzanie, czy nie istnieją jeszcze kolejki, a nawet komunikaty, których długość jest większa niż nowa wartość. Po zmianie tego atrybutu należy zrestartować aplikacje i kanały w celu upewnia się, że zmiana została uwzględniona. Nie jest możliwe wygenerowanie nowych komunikatów, które przekraczają długość MaxMsgmenedżera kolejek lub kolejki (jeśli segmentacja menedżera kolejek nie jest dozwolona).

Maksymalna długość komunikatu w definicji kanału ogranicza wielkość komunikatu, który można przesłać wzdłuż połączenia klienckiego. Jeśli aplikacja IBM MQ podejmie próbę użycia wywołania MQPUT lub wywołania 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 z klientem.

Pojęcia pokrewne

"Korzystanie z tabeli MQCONNX" na stronie 1014
Wywołania MQCONNX można użyć do określenia struktury definicji kanału (MQCD) w strukturze MQCNO.

Odsyłacze pokrewne

Maksymalna długość komunikatu (MAXMSGL)
ZMIEŃ KANAŁ
2010 (07DA) (RC2010): MQRC_DATA_LENGTH_ERROR

Wybieranie identyfikatora CCSID klienta lub serwera

Należy użyć lokalnego identyfikatora kodowanego zestawu znaków (CCSID) dla klienta. Menedżer kolejek wykonuje niezbędne konwersję. Użyj zmiennej środowiskowej MQCCSID, aby przesłonić identyfikator CCSID. Jeśli aplikacja wykonuje wiele operacji PUTs, pola CCSID i pola kodowania deskryptora MQMD mogą zostać nadpisane po zakończeniu pierwszej operacji PUT.

Dane przekazywane przez interfejs kolejki komunikatów (MQI) z aplikacji do kodu pośredniczącego klienta muszą znajdować się w lokalnym identyfikatorze CCSID, zakodowane dla IBM MQ MQI client. Jeśli połączony menedżer kolejek wymaga konwersji danych do konwersji, to konwersja jest wykonywana przez kod obsługi klienta w menedżerze kolejek.

W przypadku produktu IBM WebSphere MQ 7.0 i nowszych wersji klient Java może wykonać konwersję, jeśli menedżer kolejek nie jest w stanie wykonać tej operacji. Więcej informacji zawiera sekcja "IBM MQ classes for Java połączenia klientów" na stronie 368.

W kodzie klienta założono, że dane znakowe przekraczające interfejs MQI w kliencie są w CCSID skonfigurowanym dla tej stacji roboczej. Jeśli ten identyfikator CCSID nie jest obsługiwany lub nie jest to wymagany identyfikator CCSID, można go przesłonić za pomocą zmiennej środowiskowej MQCCSID, używając jednej z następujących komend:

- Windows

```
SET MQCCSID=850
```

- UNIX

```
export MQCCSID=850
```

- IBM i

```
ADDENVVAR ENVVAR(MQCCSID) VALUE(37)
```

Jeśli ten parametr jest ustawiony w profilu, zakłada się, że wszystkie dane MQI znajdują się na 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 identyfikatora CCSID i kodowania deskryptora MQMD są nadpisywane po zakończeniu pierwszego PUT.

Po pierwszym PUT, pola te zawierają wartość używaną przez połączonego menedżera kolejek w celu przekształcenia nagłówków IBM MQ . Upewnij się, że aplikacja zresetuje wartości do wymaganych przez nią wartości.

Użycie komendy MQINQ w aplikacji klienta

Niektóre wartości odpytywane za pomocą komendy MQINQ są modyfikowane przez kod klienta.

CCSID
jest ustawiony na identyfikator CCSID klienta, a nie identyfikator menedżera kolejek.

MaxMsgDługość
jest zmniejszony, jeśli jest ograniczony 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 sekcja MQINQ.

Korzystanie z koordynacji punktów synchronizacji w aplikacji klienckiej

Aplikacja działająca na kliencie podstawowym może wydać komendę MQCMIT i MQBACK, ale zasięg elementu sterującego punktu synchronizacji jest ograniczony do zasobów MQI. Za pomocą zewnętrznego menedżera transakcji można korzystać z rozszerzonego klienta transakcyjnego.

W produkcie IBM MQjedną z ról menedżera kolejek jest sterowanie punktem synchronizacji w ramach aplikacji. Jeśli aplikacja działa na kliencie podstawowym IBM MQ , może ona wydać komendę MQCMIT i MQBACK, ale zasięg elementu sterującego punktu synchronizacji jest ograniczony do zasobów MQI. Komenda IBM MQ MQBEGIN nie jest poprawna w podstawowym środowisku klienta.

Aplikacje działające w pełnym środowisku menedżera kolejek na serwerze mogą koordynować wiele zasobów (na przykład baz 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 użyć monitora transakcji z podstawową aplikacją kliencką.

Istnieje możliwość użycia zewnętrznego menedżera transakcji z rozszerzonym klientem transakcyjnym IBM MQ . Patrz Co to jest rozszerzony klient transakcyjny? .

Korzystanie z odczytu z wyprzedzeniem w aplikacji klienckiej

Można użyć odczytu z wyprzedzeniem na kliencie, aby zezwolić na wysyłanie nietrwałych komunikatów do klienta bez konieczności żądania przez aplikację kliencką komunikatów.

Gdy klient wymaga komunikatu z serwera, wysyła żądanie do serwera. Wysyła on osobne żądanie dla każdego z komunikatów, które konsumuje. Aby zwiększyć wydajność klientów korzystających z nietrwałych komunikatów przez uniknięcie konieczności wysyłania tych komunikatów żądań, klient

może zostać skonfigurowany do używania odczytu z wyprzedzeniem. Odczyt z wyprzedzeniem umożliwia wysyłanie komunikatów do klienta bez konieczności żądania ich żądania.

Użycie odczytu z wyprzedzeniem może zwiększyć wydajność podczas korzystania z nietrwałych komunikatów z aplikacji klienckiej. Ta poprawa wydajności jest dostępna zarówno dla aplikacji MQI, jak i aplikacji JMS . Aplikacje klienckie korzystające z wykorzystania MQGET lub asynchronicznego korzystają z ulepszeń 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:

- Klient i zdalny menedżer kolejek muszą być w wersji IBM WebSphere MQ 7.0 lub nowszej.
- 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 opcja odczytu z wyprzedzeniem jest włączona, komunikaty są wysyłane do buforu pamięci na kliencie o nazwie bufor odczytu z wyprzedzeniem. Klient ma bufor odczytu z wyprzedzeniem dla każdej kolejki, która została otwarta z włączoną obsługą odczytu z wyprzedzeniem. Komunikaty w buforze odczytu z wyprzedzeniem nie są utrwalane. Klient okresowo aktualizuje serwer, podając informacje o ilości zużywanej przez niego danych.

Nie wszystkie projekty aplikacji klienckich są odpowiednie do korzystania z odczytu z wyprzedzeniem, ponieważ nie wszystkie opcje są obsługiwane. Niektóre opcje są wymagane, aby były spójne między wywołaniami MQGET, gdy opcja odczytu z wyprzedzeniem jest włączona. Jeśli klient zmienia kryteria wyboru między wywołaniami MQGET, komunikaty zapisywane w buforze odczytu z wyprzedzeniem pozostają bez zmian w buforze odczytu z wyprzedzeniem klienta. Więcej informacji na ten temat zawiera sekcja “Zwiększanie wydajności nietrwałych komunikatów” na stronie 878.

Konfiguracja odczytu z wyprzedzeniem jest kontrolowana przez trzy atrybuty: MaximumSize, PurgeTimei UpdatePercentage, które są określone w sekcji MessageBuffer pliku konfiguracyjnego klienta IBM MQ .

Korzystanie z funkcji asynchronicznej umieszczonej w aplikacji klienckiej

Za pomocą komendy put asynchroniczny aplikacja może umieścić komunikat w kolejce bez oczekiwania na odpowiedź z menedżera kolejek. W niektórych sytuacjach można użyć tego celu w celu zwiększenia wydajności przesyłania komunikatów.

Zwykle, gdy aplikacja umieszcza komunikat lub komunikaty w kolejce za pomocą operacji MQPUT lub MQPUT1, aplikacja musi czekać na potwierdzenie przez menedżer kolejek, że przetworzyła żądanie MQI. Wydajność przesyłania komunikatów można zwiększyć, w szczególności w przypadku aplikacji, które korzystają z powiązań klienta, oraz aplikacji, które umieszczają dużą liczbę małych komunikatów w kolejce, wybierając zamiast asynchronicznie umieszczania komunikatów. Gdy aplikacja umieszcza komunikat asynchronicznie, menedżer kolejek nie zwraca powodzenia lub niepowodzenia każdego wywołania, ale można okresowo sprawdzać, czy nie wystąpiły błędy.

Aby umieścić komunikat w kolejce asynchronicznie, należy użyć opcji MQPMO_ASYNC_RESPONSE w polu *Options* w strukturze MQPMO.

Jeśli komunikat nie jest zakwalifikowany do asynchronicznego umieszczania, jest on umieszczany w kolejce synchronicznie.

W przypadku żądania asynchronicznej odpowiedzi put dla operacji MQPUT lub MQPUT1, CompCode i przyczyna MQCC_OK i MQRC_NONE nie muszą oznaczać, że komunikat został pomyślnie umieszczony w kolejce. Mimo że powodzenie lub niepowodzenie każdej pojedynczej wywołania MQPUT lub MQPUT1 może nie zostać zwrócone natychmiast, pierwszy błąd, który wystąpił w wywołaniu asynchronicznym, można określić później za pomocą wywołania MQSTAT.

Więcej informacji na temat opcji MQPMO_ASYNC_RESPONSE zawiera sekcja Opcje MQPMO.

Przykładowy program Asynchroniczny Put demonstruje niektóre z dostępnych funkcji. Szczegółowe informacje na temat funkcji i projektu programu oraz sposobu jego uruchamiania zawiera sekcja “Przykładowy program do umieszczania w pamięci asynchronicznej” na stronie 1185.

Korzystanie z współużytkowania konwersacji w aplikacji klienckiej

W środowisku, w którym dozwolone jest współużytkowanie konwersacji, konwersacje mogą współużytkować instancję kanału MQI.

Współużytkowanie konwersacji jest kontrolowane przez dwa pola o nazwie SharingConversations, z których jeden jest częścią struktury definicji kanału (MQCD), a jeden z nich jest częścią struktury parametru wyjścia kanału (MQCXP). Pole SharingConversations w tabeli MQCD jest liczbą całkowitą, określającą maksymalną liczbę konwersacji, które mogą współużytkować instancję kanału powiązaną z kanałem. Pole SharingConversations w tabeli MQCXP jest wartością boolowką wskazującą, czy instancja kanału jest obecnie współużytkowana.

W środowisku, w którym współużytkowanie konwersacji nie jest dozwolone, nowe połączenia klienckie określające identyczne tabele MQCD nie będą współużytkować instancji kanału.

Nowe połączenie aplikacji klienckiej będzie współużytkować instancję kanału, gdy spełnione są następujące warunki:

- Zarówno połączenie klienta, jak i połączenia z serwerem kończą się instancją kanału, są skonfigurowane do współużytkowania konwersacji, a te wartości nie są nadpisywane przez wyjścia kanału.
- Wartość MQCD połączenia klienckiego (podana w wywołaniu klienta MQCONNX lub z tabeli definicji kanału klienta (CCDT)) jest dokładnie zgodna z wartością MQCD połączenia klienta podaną w wywołaniu MQCONNX klienta lub z tabeli definicji kanału klienta, gdy istniejąca instancja kanału została najpierw utworzona. Należy zauważyć, że oryginalna tabela MQCD mogła zostać później zmieniona przez wyjścia lub negocjację kanału, ale zgodność ta jest porównana z wartością dostarczonej do systemu klienta, zanim zmiany te zostały wprowadzone.
- Limit współużytkowania konwersacji po stronie serwera nie został przekroczony.

Jeśli nowe połączenie aplikacji klienckiej jest zgodne z kryteriami umożliwiania współużytkowania instancji kanału z innymi konwersacjami, ta decyzja jest podejmowana przed wywołaniem jakichkolwiek wyjść w tej konwersacji. Wyjścia w takiej konwersacji nie mogą zmienić faktu, że współużytkuje instancję kanału z innymi konwersacjami. Jeśli nie ma istniejących instancji kanałów zgodnych z nową definicją kanału, nawiąże połączenie z nową instancją kanału.

Negocjowanie kanału jest wykonywane tylko dla pierwszej konwersacji w instancji kanału; wynegocjowane wartości dla instancji kanału są ustalone na tym etapie i nie można ich zmieniać podczas kolejnych konwersacji. Uwierzytelnianie TLS występuje również tylko w przypadku pierwszej konwersacji.

Jeśli wartość parametru MQCD SharingConversations zostanie zmieniona podczas inicjowania dowolnego zabezpieczenia, wysłania lub odebrania wyjścia dla pierwszej konwersacji w gnieździe przy użyciu połączenia klienckiego lub końca połączenia z serwerem instancji kanału, nowa wartość, która ma po zainicjowaniu wszystkich tych wyjść, jest używana do określenia wartości konwersacji współużytkowania dla instancji kanału (pierwszeństwo ma najniższa wartość).

Jeśli wynegocjowana wartość współużytkowania konwersacji wynosi zero, instancja kanału nigdy nie jest współużytkowana. Dalsze programy obsługi wyjścia, które ustawiają to pole na zero w podobny sposób, działają w swojej własnej instancji kanału.

Jeśli wynegocjowana wartość współużytkowania konwersacji jest większa niż zero, wówczas parametr MQCXP SharingConversations ma wartość TRUE w przypadku kolejnych wywołań wyjść, wskazując, że inne programy obsługi wyjścia w tej instancji kanału mogą być wprowadzane równocześnie z tym programem.

Podczas pisania programu obsługi wyjścia kanału należy rozważyć, czy będzie on uruchamiany w instancji kanału, która może obejmować współużytkowanie konwersacji. Jeśli instancja kanału może obejmować współużytkowanie konwersacji, należy wziąć pod uwagę wpływ na inne instancje wyjścia kanału zmiany pól MQCD. Wszystkie pola MQCD mają wspólne wartości we wszystkich konwersacjach współużytkowania. Po nawiązaniu instancji kanału, jeśli programy obsługi wyjścia próbują zmienić zawartość 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 zmaterializowanej tabeli MQCD w kodzie wyjścia.

W przypadku pracy z kanałem zdefiniowanym w celu współużytkowania konwersacji, ale nie ma potrzeby współużytkowania dla konkretnej instancji kanału, należy ustawić wartość parametru MQCD SharingConversations na wartość 1 lub 0 po zainicjowaniu wyjścia kanału w pierwszej konwersacji w instancji kanału. Informacje na temat wartości SharingConversationszawiera sekcja SharingConversations .

Przykład

Współużytkowanie konwersacji jest włączone.

Korzystasz z definicji kanału połączenia klienckiego, która określa program obsługi wyjścia.

Przy pierwszym uruchomieniu tego kanału program obsługi wyjścia zmienia niektóre parametry MQCD po jego zainicjowaniu. Są one wykonywane przez kanał, więc definicja, z którą jest uruchomiony kanał, różni się od tej, która została pierwotnie dostarczona. Parametr MQCXP SharingConversations jest ustawiony na wartość TRUE.

Następnym razem, gdy aplikacja nawiąże połączenie przy użyciu tego kanału, konwersacja zostanie uruchomiona w instancji kanału, która została wcześniej uruchomiona, ponieważ ma ona tę samą definicję kanału oryginalnego. Instancja kanału, z którą aplikacja nawiązuje połączenie po raz drugi, jest tą samą instancją, co po raz pierwszy, z którą jest połączona. W związku z tym korzysta z definicji, które zostały zmienione przez program obsługi wyjścia. Gdy program obsługi wyjścia jest inicjowany dla drugiej konwersacji, mimo że może zmieniać pola MQCD, nie są one wykonywane przez kanał. Te same charakterystyki mają zastosowanie do wszystkich kolejnych konwersacji, które współużytkują instancję kanału.

Korzystanie z tabeli MQCONNX

Wywołania MQCONNX można użyć do określenia struktury definicji kanału (MQCD) w strukturze MQCNO.

Umożliwia to wywoływanie aplikacji klienckiej w celu określenia definicji kanału połączenia klienckiego w czasie wykonywania. Więcej informacji na ten temat zawiera sekcja Korzystanie z struktury MQCNO w wywołaniu MQCONNX. Jeśli używany jest produkt MQCONNX, wywołanie wysłane na serwerze zależy od konfiguracji poziomu serwera i procesu nasłuchującego.

W przypadku korzystania z klienta MQCONNX z poziomu klienta następujące opcje są ignorowane:

- MQCNO_STANDARD_BINDING
- MQCNO_FASTPATH_BINDING

Struktura MQCD, której można użyć, zależy od numeru wersji produktu MQCD, który jest używany. Informacje na temat wersji MQCD (MQCD_VERSION) zawiera sekcja MQCD Version(Wersja MQCD). Struktury MQCD można użyć na przykład do przekazywania programów obsługi wyjścia kanału na serwer. Jeśli używany jest produkt MQCD w wersji 3 lub nowszej, można użyć struktury do przekazania tablicy wyjść do serwera. Tej funkcji można użyć do wykonania więcej niż jednej operacji w tym samym komunikacie, na przykład do szyfrowania i kompresji, przez dodanie wyjścia dla każdej operacji, zamiast modyfikowania istniejącego wyjścia. Jeśli w strukturze MQCD nie zostanie określona tablica, sprawdzane będą pojedyncze pola wyjścia. Aby uzyskać więcej informacji na temat programów obsługi wyjścia kanału, patrz “Programy obsługi wyjścia kanału dla kanałów przesyłania komunikatów” na stronie 1059.

Uchwyty połączeń współużytkowanych na tabeli MQCONNX

Istnieje możliwość współużytkowania uchwytów między różnymi wątkami w ramach tego samego procesu, przy użyciu uchwytów połączeń współużytkowanych.

W przypadku określenia uchwytu połączenia współużytkowanego uchwyt połączenia zwrócony z wywołania MQCONNX może być przekazywany w kolejnych wywołaniach MQI w dowolnym wątku w procesie.

Uwaga: Za pomocą uchwytu połączenia współużytkowanego na serwerze IBM MQ MQI client można połączyć się z menedżerem kolejek serwera, który nie obsługuje uchwytów połączeń współużytkowanych.

Więcej informacji na ten temat zawiera sekcja “Korzystanie z tabeli MQCONNX” na stronie 1014.

Budowanie aplikacji dla produktu IBM MQ MQI clients

Aplikacje mogą być budowane i uruchamiane w środowisku IBM MQ MQI client . Aplikacja musi być zbudowana i dowiązana do używanej partycji IBM MQ MQI client . Sposób budowania i tworzenia powiązanych aplikacji różni się w zależności od używanej platformy i języka programowania.

Jeśli aplikacja ma być uruchamiana w środowisku klienta, można ją zapisać w językach przedstawionych w poniższej tabeli:

Tabela 137. Języki programowania obsługiwane w środowiskach klienckich

Platforma klienta	C	C++	COBOL	pTAL	RPG	Visual Basic
AIX	Tak	Tak	Tak			
IBM i	Tak		Tak		Tak	
Linux	Tak	Tak	Tak			
Solaris	Tak	Tak	Tak			
Windows	Tak	Tak	Tak			Tak

Multi *Łączenie aplikacji C z kodem produktu IBM MQ MQI client*

Po zapisaniu aplikacji IBM MQ , która ma być uruchamiana na serwerze IBM MQ MQI client, należy połączyć ją z kodem produktu IBM MQ MQI client .

Istnieje możliwość połączenia aplikacji z kodem produktu IBM MQ MQI client na dwa sposoby:

1. Bezpośrednio, łącząc aplikację z menedżerem kolejek, w takim przypadku menedżer kolejek musi znajdować się na tym samym komputerze, co aplikacja użytkownika.
2. Do pliku biblioteki klienta, który zapewnia dostęp do menedżerów kolejek na tym samym lub na innym komputerze.

Produkt 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.

Solaris **Solaris**

libmqic.so.

Aby użyć programów na komputerze, na którym jest zainstalowany tylko IBM MQ MQI client for Solaris , należy zrekompilować programy w celu połączenia ich z biblioteką klienta:

```
$ /opt/SUNWspro/bin/cc -o prog_name prog_name c -mt -lmqic \
-lsocket -lc -lnsl -ldl
```

Parametry muszą zostać wprowadzone w poprawnej kolejności, tak jak pokazano poniżej.

Windows **Windows**
MQIC32.LIB.

ULW *Łączenie aplikacji C++ z kodem IBM MQ MQI client*

Można pisać aplikacje, które mają być uruchamiane na kliencie w języku C + +. Metody budowania różnią się w zależności od środowiska.

Informacje na temat łączenia aplikacji w języku C++ można znaleźć w sekcji Budowanie programów IBM MQ C++.

Szczegółowe informacje na temat wszystkich aspektów używania języka C + + zawiera sekcja Używanie języka C++ .

Multi *Łączenie aplikacji w języku COBOL z kodem produktu IBM MQ MQI client*

Po napisanie aplikacji w języku COBOL, która ma być uruchamiana na serwerze IBM MQ MQI client, należy połączyć ją z odpowiednią biblioteką.

Produkt IBM MQ udostępnia plik biblioteki klienta dla każdego środowiska:

AIX **AIX**
Powiąż niewielowątkową aplikację COBOL z biblioteką libmqicb.a lub wielowątkową aplikacją COBOL z biblioteką libmqicb_r.a.

IBM i **IBM i**
Powiąż aplikację kliencką COBOL z programem usługowym AMQCSTUB dla aplikacji niewielowątkowych lub program usługowy AMQCSTUB_R dla aplikacji wielowątkowych.

Solaris **Solaris**
Powiąż niewielowątkową aplikację w języku COBOL z biblioteką libmqicb.so lub z wątkiem aplikacji COBOL z biblioteką libmqicb_r.so.

Windows **Windows**
Powiąż kod aplikacji z biblioteką MQICCBB dla 32-bitowego języka COBOL. Produkt IBM MQ MQI client for Windows nie obsługuje 16-bitowego języka COBOL.

Windows *Łączenie aplikacji produktu Visual Basic z kodem produktu IBM MQ MQI client*

Aplikacje produktu Microsoft Visual Basic można połączyć z kodem IBM MQ MQI client w systemie Windows.

W produkcie IBM MQ 9.0obsługa produktu IBM MQ dla produktu Microsoft Visual Basic 6.0 jest nieaktualna. Zalecaną technologią zastępczą są klasy IBM MQ dla .NET. Więcej informacji na ten temat zawiera sekcja Tworzenie aplikacji .NET.

Połącz aplikację Visual Basic z następującymi plikami włączenie:

CMQB.bas
MQI

CMQBB.bas
MQAI

CMQCFB.bas
Komendy PCF

CMQXB.bas
Kanały

Ustaw `mqtype=2` dla klienta w kompilatorze Visual Basic , aby upewnić się, że automatyczny wybór biblioteki dll klienta jest poprawny:

MQIC32.dll
Windows 7, Windows 8, Windows 2008 i Windows 2012

Pojęcia pokrewne

“Kodowanie w Visual Basic” na stronie 1160
Informacje, które należy wziąć pod uwagę podczas kodowania programów IBM MQ w programie Microsoft Visual Basic. Produkt Visual Basic jest obsługiwany tylko w systemie Windows.

“Przygotowywanie programów Visual Basic w programie Windows” na stronie 1126
Informacje do rozważenia podczas korzystania z 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 “Odwołanie do definicji połączenia przy użyciu wyjścia wstępnego z połączeniem z repozytorium” na stronie 1092.
2. Jeśli zmienna środowiskowa MQSERVER jest ustawiona, używany jest zdefiniowany przez nią kanał.
3. Jeśli plik `mqclient.ini` jest zdefiniowany i zawiera parametry ServerConnection, używany jest zdefiniowany przez niego kanał. Więcej informacji na ten temat zawiera sekcja Konfigurowanie klienta przy użyciu pliku konfiguracyjnego i sekcja CHANNELS w pliku konfiguracyjnym klienta.
4. Jeśli zmienne środowiskowe MQCHLLIB i MQCHLTAB są ustawione, używana jest tabela definicji kanału klienta, którą wskazują. Alternatywnie, począwszy od IBM MQ 9.0, zmienna środowiskowa MQCCDTURL udostępnia możliwość ustawienia kombinacji zmiennych środowiskowych MQCHLLIB i MQCHLTAB. Jeśli parametr MQCCDTURL jest ustawiony, używana jest tabela definicji kanału klienta, na którą wskazuje. Więcej informacji na ten temat zawiera sekcja Dostęp do tabeli definicji kanału klienta z możliwością adresowania w sieci WWW.

5. Jeśli plik `mqclient.ini` jest zdefiniowany i zawiera katalog ChannelDefinitionoraz atrybuty pliku ChannelDefinition, atrybuty te są używane do znajdowania tabeli definicji kanału klienta. Więcej informacji na ten temat zawiera sekcja Konfigurowanie klienta przy użyciu pliku konfiguracyjnego i sekcja CHANNELS 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 ustanowione na podstawie wartości `DefaultPrefix` w pliku `mqs.ini` . Jeśli wyszukiwanie tabeli definicji kanału klienta nie powiedzie się, klient użyje następujących ścieżek:
   - 

W systemie UNIX and Linux: `/var/mqm/AMQCLCHL.TAB`
   - 

W systemie Windows: `C:\Program Files\IBM\MQ\amqclchl.tab`
   - 

W systemie IBM i: `/QIBM/UserData/mqm/@ipcc`
   - 

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

Tabela definicji kanału klienta

Dostęp WWW do tabeli definicji kanału klienta

Zadania pokrewne

Konfigurowanie połączeń między serwerem i klientem

Odsyłacze pokrewne

SERWER MQSERVER

Biblioteka MQCHLLIB

MQCHLTAB

Adres MQCCDTURL

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 służy do definiowania 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 produkcie IBM MQ 9.0zmienna środowiskowa MQCCDTURL udostępnia możliwość ustawienia kombinacji zmiennych środowiskowych MQCHLLIB i MQCHLTAB. Opcja MQCCDTURL umożliwia podanie

pliku, adresu FTP lub adresu URL protokołu HTTP 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 z użyciem struktury MQCNO wywołania MQCONNX.

Więcej informacji na ten temat zawiera sekcja Korzystanie ze struktury MQCNO w wywołaniu MQCONNX.

Łączenie aplikacji klienckich z menedżerami kolejek przy użyciu tabeli definicji kanału klienta
Jeśli komenda MQSC DEFINE CHANNEL zostanie użyta, szczegóły wprowadzone przez użytkownika zostaną umieszczone w tabeli definicji kanału klienta (ccdt). Zawartość parametru **QMgrName** wywołania MQCONN lub MQCONNX określa menedżera kolejek, z którym łączy się klient.

Dostęp do tego pliku jest uzyskiwany przez klienta w celu określenia kanału, który będzie używany przez aplikację. Jeśli istnieje więcej niż jedna odpowiednia definicja kanału, wybór kanału ma wpływ na atrybuty kanału klienta wagi kanału (CLNTWGHT) i powinowactwa połączenia (AFFINITY).

Korzystanie z automatycznego ponownego połączenia klienta
Aplikacje klienckie mogą być automatycznie ponownie połączone, bez konieczności pisania dodatkowego kodu, poprzez skonfigurowanie wielu 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 połączenie 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 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 .

Aplikacja kliencka może używać wielu sposobów korzystania z tabeli definicji kanału klienta. Tabelę CCDT można skopiować na komputer kliencki. Istnieje możliwość skopiowania tabeli definicji kanału klienta do położenia współużytkowanego przez więcej niż jednego klienta. Pakiet CCDT można udostępnić klientowi jako plik współużytkowany, natomiast jego położenie znajduje się na serwerze.

W produkcie IBM MQ 9.0pakiet CCDT może być udostępniany w centralnej lokalizacji, która jest dostępna za pośrednictwem identyfikatora URI, co powoduje usunięcie konieczności indywidualnego aktualizowania tabeli definicji kanału klienta dla każdego wdrożonego klienta.

Pojęcia pokrewne
Tabela definicji kanału klienta
Dostęp do tabeli definicji kanału klienta w sieci WWW
Zadania pokrewne
Uzyskiwanie dostępu do definicji kanału połączenia klienckiego

Grupy menedżerów kolejek w tabeli definicji kanału klienta
Istnieje możliwość zdefiniowania zestawu połączeń w tabeli definicji kanału klienta (CCDT) jako *grupy menedżerów kolejek*. Istnieje możliwość połączenia aplikacji z menedżerem kolejek, który jest częścią grupy menedżerów kolejek. Można to zrobić, dodając przedrostek nazwy menedżera kolejek w wywołaniu programu MQCONN lub MQCONNX z gwiazdką.

Można wybrać opcję definiowania połączeń z więcej niż jednym serwerem, ponieważ:

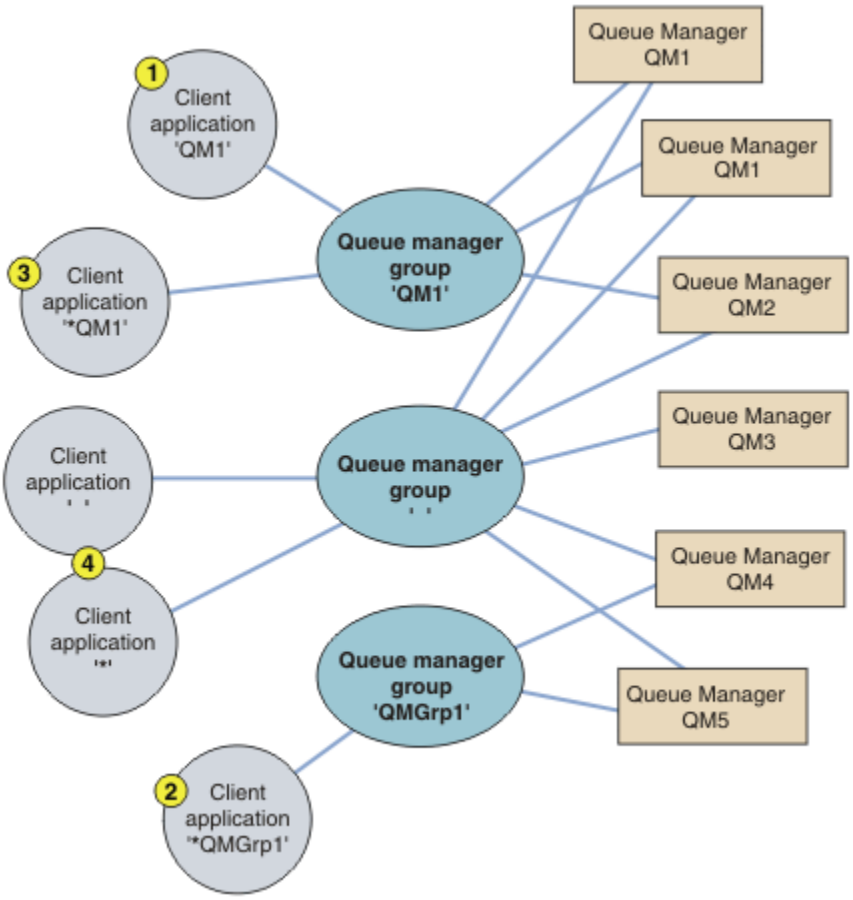
- Klient ma zostać podłączony do dowolnego zestawu menedżerów kolejek, który jest uruchomiony, w celu zwiększenia dostępności.
- Użytkownik chce ponownie połączyć klienta z tym samym menedżerem kolejek, który został pomyślnie połączony z tym samym menedżerem kolejek, ale nawiąże połączenie z innym menedżerem kolejek, jeśli połączenie nie powiedzie się.
- Jeśli połączenie nie powiedzie się, należy ponowić próbę nawiązania połączenia klienta z innym menedżerem kolejek. W tym celu należy ponownie wydać komendę `MQCONN` w programie klienckim.
- Jeśli połączenie nie powiedzie się, użytkownik chce automatycznie ponownie połączyć połączenie klienta z innym menedżerem kolejek bez zapisywania kodu klienta.
- Użytkownik chce automatycznie ponownie połączyć połączenie klienta z inną instancją menedżera kolejek z wieloma instancjami, jeśli instancja rezerwowa przejmuje tę instancję, bez zapisywania kodu klienta.
- Użytkownik chce zrównoważyć połączenia klientów w wielu menedżerach kolejek, przy czym więcej klientów łączy się z niektórymi menedżerami kolejek niż inne.
- Użytkownik chce rozpowszechnić ponowne połączenie wielu połączeń klienckich w wielu menedżerach kolejek i w czasie, w przypadku gdy duża liczba połączeń powoduje niepowodzenie.
- Użytkownik chce mieć możliwość przeniesienia menedżerów kolejek bez zmiany kodu aplikacji klienckiej.
- Użytkownik chce napisać programy użytkowe klienta, które nie muszą znać nazw menedżerów kolejek.

Łączenie się z różnymi menedżerami kolejek nie zawsze jest odpowiednie. Na przykład rozszerzony klient transakcyjny lub klient Java w produkcie WebSphere Application Servermoże wymagać nawiązania 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 to zestaw połączeń zdefiniowanych w tabeli definicji kanału klienta (CCDT). Zestaw jest zdefiniowany przez jego członków o tej samej wartości atrybutu **QMNAME** w definicjach kanałów.

Rysunek 102 na stronie 1021 jest graficzną reprezentacją tabeli połączeń klienta, przedstawiając trzy grupy menedżerów kolejek, dwie nazwane grupy menedżerów kolejek zapisane w tabeli CCDT jako **QMNAME** (`QM1`) i **QMNAME** (`QMGrp1`), a także jedną pustą lub domyślną grupę napisaną jako **QMNAME** (`''`).

1. Grupa menedżerów kolejek `QM1` ma trzy kanały połączenia klienckiego, łącząc je z menedżerami kolejek `QM1` i `QM2`. `QM1` może być menedżerem kolejek z wieloma instancjami, który znajduje się na dwóch różnych serwerach.
2. Domyślna grupa menedżerów kolejek ma sześć kanałów połączenia klienckiego łączących go ze wszystkimi menedżerami kolejek.
3. `QMGrp1` ma kanały połączenia klienta z dwoma menedżerami kolejek, `QM4` i `QM5`.



Rysunek 102. Grupy menedżerów kolejek

Cztery przykłady użycia tej tabeli połączeń klienta są opisane przy pomocy numerowanych aplikacji klienckich w produkcie Rysunek 102 na stronie 1021.

1. W pierwszym przykładzie aplikacja kliencka przekazuje nazwę menedżera kolejek, `QM1`, jako parametr **`QmgrName`** do jej wywołania MQI produktu `MQCONN` lub `MQCONNX` . Kod klienta IBM MQ wybiera zgodną grupę menedżerów kolejek, QM1. Grupa zawiera trzy kanały połączenia, a program IBM MQ MQI client próbuje połączyć się z serwerem `QM1` za pomocą każdego z tych kanałów, dopóki nie znajdzie programu nasłuchującego IBM MQ dla połączenia przyłączonego do działającego menedżera kolejek o nazwie `QM1`.

   Kolejność prób połączenia zależy od wartości atrybutu AFFINITY połączenia klienckiego i współczynników korygujący kanału klienta. W ramach tych ograniczeń kolejność prób połączeń jest losowa, zarówno w przypadku trzech możliwych połączeń, jak i w czasie, w celu rozłożenia obciążenia nawiązając połączenia.

   Wywołanie `MQCONN` lub `MQCONNX` wydane przez aplikację kliencką powiedzie się, gdy połączenie zostanie nawiązane z działającą instancją serwera `QM1`.

2. W drugim przykładzie aplikacja kliencka przekazuje nazwę menedżera kolejek poprzedzoną gwiazdką (`*QMGrp1` jako parametr **`QmgrName`** do jej wywołania MQI produktu `MQCONN` lub `MQCONNX`). Klient IBM MQ wybiera zgodną grupę menedżerów kolejek QMGrp1. Ta grupa zawiera dwa kanały połączenia klienckiego, a program IBM MQ MQI client próbuje połączyć się z *dowolnym* menedżerem kolejek przy użyciu każdego kanału z kolei. W tym przykładzie program IBM MQ MQI client musi nawiązać pomyślnie połączenie. Nazwa menedżera kolejek, z którym łączy się połączenie, nie ma znaczenia.

   Reguła dla kolejności prób nawiązania połączenia jest taka sama jak poprzednio. Jedyna różnica polega na tym, że poprzedzając nazwę menedżera kolejek gwiazdką, klient wskazuje, że nazwa menedżera kolejek nie jest istotna.

Wywołanie MQCONN lub MQCONNX wystawione przez aplikację kliencką powiedzie się, gdy połączenie zostanie nawiązane z działającą instancją dowolnego menedżera kolejek połączonego z kanałami w grupie menedżerów kolejek QMGrp1 .

3. Trzeci przykład jest zasadniczo taki sam, jak drugi, ponieważ parametr **QmgrName** jest poprzedzony gwiazdką (*QM1). Przykład ten ilustruje, że nie można określić, który menedżer kolejek połączenia kanału klienta ma nawiązać połączenie, sprawdzając atrybut QMNAME w jednej definicji kanału przez siebie. Fakt, że atrybut **QMNAME** definicji kanału to QM1, nie jest wystarczający do tego, aby było możliwe nawiązanie połączenia z menedżerem kolejek o nazwie QM1. Jeśli aplikacja kliencka prefikuje swój parametr **QmgrName** z gwiazdką, to dowolny menedżer kolejek może być elementem docelowym połączenia.

   W tym przypadku wywołania MQCONN lub MQCONNX wydane przez aplikację kliencką powiodą się, gdy połączenie zostanie nawiązane z działającą instancją klasy QM1 lub QM2.

4. W czwartym przykładzie przedstawiono użycie grupy domyślnej. W takim przypadku aplikacja kliencka przekazuje gwiazdkę, '*'lub pustą ' ', jako parametr **QmgrName** do jej wywołania MQI produktu MQCONN lub MQCONNX . Zgodnie z konwencją w definicji kanału klienta, pusty atrybut **QMNAME** oznacza domyślną grupę menedżerów kolejek, a pusty lub gwiazdka parametru **QmgrName** jest zgodny z pustym atrybutem **QMNAME** .

   W tym przykładzie domyślna grupa menedżerów kolejek ma połączenia kanału klienta ze wszystkimi menedżerami kolejek. Wybierając domyślną grupę menedżerów kolejek, aplikacja może być połączona z dowolnym menedżerem kolejek w grupie.

   Wywołanie MQCONN lub MQCONNX wystawione przez aplikację kliencką powiedzie się, gdy połączenie zostanie nawiązane z działającą instancją dowolnego menedżera kolejek.

   Uwaga: Grupa domyślna różni się od domyślnego menedżera kolejek, mimo że aplikacja używa pustego parametru **QmgrName** do łączenia się z domyślną grupą menedżerów kolejek lub do domyślnego menedżera kolejek. Pojęcie domyślnej grupy menedżerów kolejek ma znaczenie tylko w przypadku aplikacji klienckiej, a domyślny menedżer kolejek jest odpowiedni dla aplikacji serwera.

Zdefiniuj kanały połączenia klienckiego tylko w jednym menedżerze kolejek, w tym te kanały, 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 tego tematu. W jaki sposób korzystanie z grupy menedżerów kolejek udostępnia te możliwości?

Nawiąże połączenie z dowolnym zestawem menedżerów kolejek.
Zdefiniuj grupę menedżerów kolejek z połączeniami do wszystkich menedżerów kolejek w zestawie i nawiąże połączenie z grupą za pomocą parametru **QmgrName** z przedrostkiem gwiazdki.

Ponownie nawiąże połączenie z tym samym menedżerem kolejek, ale nawiąże połączenie z innym menedżerem kolejek, jeśli menedżer kolejek połączony z ostatnim czasem jest niedostępny.
Zdefiniuj grupę menedżerów kolejek tak, jak wcześniej, ale ustaw atrybut **AFFINITY** (PREFEROWANE) dla każdej definicji kanału klienta.

Jeśli połączenie nie powiedzie się, ponów próbę nawiązania połączenia z innym menedżerem kolejek.
Nawiąże połączenie z grupą menedżerów kolejek i ponownie wydaj wywołanie MQI produktu MQCONN lub MQCONNX , jeśli połączenie zostało zerwane lub menedżer kolejek nie powiedzie się.

Jeśli połączenie nie powiedzie się, automatycznie ponownie nawiąże połączenie z innym menedżerem kolejek.
Połącz się z grupą menedżerów kolejek za pomocą opcji MQCONNX **MQCNO** MQCNO_RECONNECT.

Automatycznie ponownie nawiąże połączenie z inną instancją menedżera kolejek z wieloma instancjami.
Wykonaj to samo, 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 z instancjami menedżera kolejek z wieloma instancjami.

Można również poprosić aplikację kliencką o wywołanie jej wywołania MQI produktu MQCONN lub MQCONNX bez znaku gwiazdki poprzedzonej przedrostkiem w parametrze **QmgrName** . W ten sposób aplikacja kliencka może połączyć się tylko z nazwanym 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. Tej wartości można również użyć do ograniczenia ponownego połączenia z tą samą instancją zwykłego menedżera kolejek.

Równoważenie połączeń klientów między menedżerami kolejek, z większą liczbę klientów połączonych z niektórymi menedżerami kolejek niż inne.
Zdefiniuj grupę menedżerów kolejek i ustaw atrybut **CLNTWGHT** dla każdej definicji kanału klienta, aby nierównomiernie rozprowadzać połączenia.

Rozłożyć ponownie obciążenie ponownie połączenia klienta i rozłożyć je z upływem czasu, po awarii połączenia lub menedżera kolejek.
Wykonaj to samo, co w poprzednim przykładzie. IBM MQ MQI client losuje ponowne połączenia między menedżerami kolejek i rozprzestrzenia ponowne połączenia w czasie.

Przenieś menedżery kolejek bez zmiany kodu klienta.
Pakiet CCDT izoluje aplikację kliencką z miejsca, w którym znajduje się menedżer kolejek. Pakiet 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 organizacji, a także odzwierciedlać architekturę rozwiązań zamiast nadawania nazw menedżerom kolejek.

z/OS *Łączenie z grupami współużytkowania kolejek*

Istnieje możliwość połączenia aplikacji z menedżerem kolejek, który jest częścią grupy współużytkowania kolejek. Można to zrobić, korzystając z nazwy grupy współużytkowania kolejki 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 kolejki do łączenia się z dostępnym menedżerem kolejek w grupie. Więcej informacji na ten temat zawiera sekcja Podłączanie klienta do grupy współużytkowania kolejek. Należy sprawdzić, 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 ważenia kanałów i powinowactwa

Poniższe przykłady ilustrują sposób, w jaki kanały połączenia klienckiego są wybierane, gdy używane są niezerowe wartości ClientChannelWeights .

Atrybuty kanału ClientChannelWeight (Waga) i ConnectionAffinity (ConnectionAffinity) sterują sposobem wyboru kanałów połączenia klienckiego, gdy dla połączenia jest dostępny więcej niż jeden odpowiedni kanał. Kanały te są skonfigurowane do łączenia się z różnymi menedżerami kolejek w celu zapewnienia wyższej dostępności, równoważenia obciążenia lub obu tych funkcji. Wywołania MQCONN, które mogą być wynikiem połączenia z jednym z kilku menedżerów kolejek, muszą poprzedzić nazwę menedżera kolejek gwiazdką zgodnie z opisem w: Przykłady wywołań MQCONN: przykład 1. Nazwa menedżera kolejek zawiera znak gwiazdki (*).

Odpowiednie kanały kandydujące dla połączenia to te, w których atrybut QMNAME jest zgodny z nazwą menedżera kolejek określoną w wywołaniu MQCONN. Jeśli wszystkie mające zastosowanie kanały dla połączenia mają wartość ClientChannelWaga równą zeru (wartość domyślna), to są one wybierane w kolejności alfabetycznej, tak jak w przykładzie: Przykłady wywołań MQCONN: Przykład 1. Nazwa menedżera kolejek zawiera znak gwiazdki (*).

Poniższe przykłady ilustrują, co się dzieje, gdy używane są niezerowe wartości `ClientChannelWeights` . Należy zauważyć, że ponieważ ta funkcja obejmuje pseudo-losowy wybór kanału, przykłady pokazują sekwencję działań, które mogą się zdarzyć, a nie to, co na pewno się da.

Przykład 1. Wybieranie kanałów, gdy właściwość ConnectionAffinity jest ustawiona na wartość PREFERRED. W tym przykładzie pokazano, w jaki sposób IBM MQ MQI client wybiera kanał z tabeli definicji kanału klienta, gdzie wartość ConnectionAffinity jest ustawiona na wartość PREFERRED.

W tym przykładzie kilka maszyn klienckich korzysta z tabeli definicji kanału klienta (CCDT) udostępnianej przez menedżer kolejek. Pakiet CCDT zawiera kanały połączeń klientów z następującymi atrybutami (pokazywane 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)
```

Problemy z aplikacją MQCONN (*CORE)

Kanał A nie jest kandydatem do tego połączenia, ponieważ atrybut QMNAME nie jest zgodny. Kanały B, C i D są identyfikowane jako kandydaci i są umieszczane w kolejności preferencji w oparciu o ich wagę. W tym przykładzie zamówienie może mieć wartość C, B, D. Klient próbuje połączyć się z menedżerem kolejek w pliku core2.ops.company.example. Nazwa menedżera kolejek pod tym adresem nie jest sprawdzona, ponieważ wywołanie MQCONN zawierło gwiazdkę w nazwie menedżera kolejek.

Należy pamiętać, że przy użyciu produktu `AFFINITY(PREFERRED)` za każdym razem, gdy ten konkretny klient łączy się z tym klientem, kanały będą umieszczać w tej samej kolejności początkowej preferencji. Ma to zastosowanie nawet wtedy, gdy połączenia pochodzą z różnych procesów lub w różnych momentach.

W tym przykładzie nie można uzyskać dostępu do menedżera kolejek na poziomie core.2.ops.company.example . Klient próbuje połączyć się z plikiem core1.ops.company.example , ponieważ kanał B jest następny w kolejności preferencji. Ponadto kanał C został zdegrazowany, aby stać się najmniej preferowanym.

Drugie wywołanie MQCONN (*CORE) jest wydawane przez tę samą aplikację. Kanał C został zdegrazowany przez poprzednie połączenie, więc najbardziej preferowanym kanałem jest teraz B. To połączenie jest nawiązane z plikiem core1.ops.company.example.

Druga maszyna współużytkuje tę samą tabelę definicji kanału klienta, co umożliwia umieszczenie kanałów w innej kolejności początkowej. Na przykład D, B, C. W normalnych okolicznościach, ze wszystkimi kanałami pracujące, aplikacje na tym komputerze są połączone z plikiem core3.ops.company.example , podczas gdy te na pierwszym komputerze są połączone z plikiem core2.ops.company.example. Pozwala to na równoważenie obciążenia dużej liczby klientów w wielu menedżerach kolejek, pozwalając każdemu klientowi na nawiązanie połączenia z tym samym menedżerem kolejek, jeśli jest on dostępny.

Przykład 2. Wybieranie kanałów, gdy parametr ConnectionAffinity jest ustawiony na wartość NONE. W tym przykładzie pokazano, w jaki sposób IBM MQ MQI client wybiera kanał z tabeli definicji kanału klienta, gdzie wartość ConnectionAffinity jest ustawiona na wartość NONE.

W tym przykładzie pewna liczba klientów korzysta z tabeli definicji kanału klienta (CCDT) udostępnianej przez menedżera kolejek. Pakiet CCDT zawiera kanały połączeń klientów z następującymi atrybutami (pokazywane 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 wydaje MQCONN (*CORE). Podobnie jak w poprzednim przykładzie, kanał A nie jest uwzględniany, 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 może zostać wybrany kanał B. Nie utworzono trwałej kolejności preferencji.

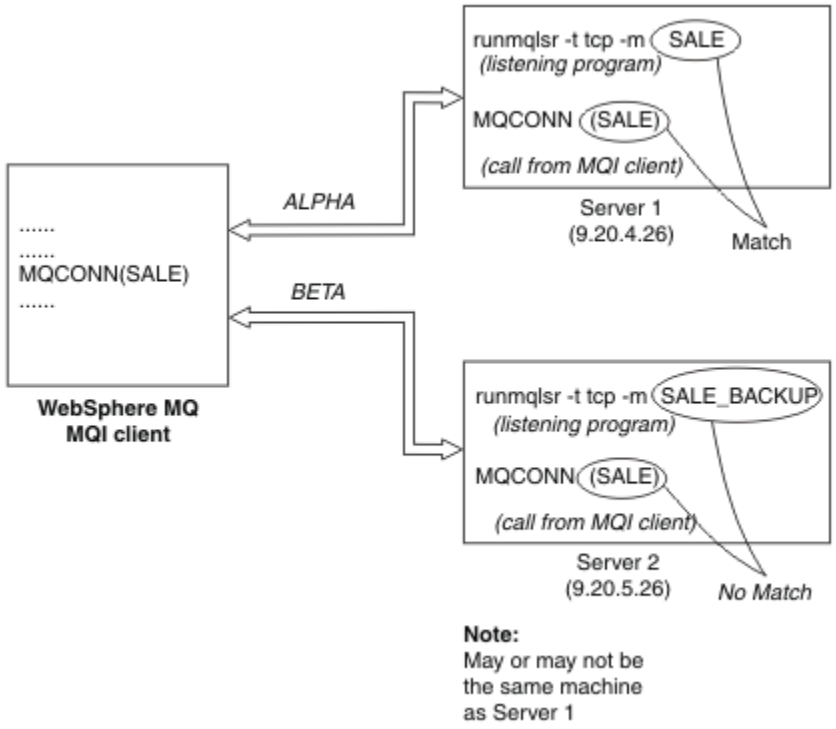
Zostanie wykonane drugie wywołanie MQCONN (*CORE). Ponownie wybierany jest jeden z trzech odpowiednich kanałów, z takimi samymi prawdopodobieństwami. W tym przykładzie wybierany jest kanał C. Jednak plik core2.ops.company.example nie odpowiada, dlatego między pozostałymi kanałami kandydującymi jest dokonany inny wybór. Wybrany jest kanał B, a aplikacja jest połączona z plikiem core1.ops.company.example.

W przypadku wartości AFFINITY (NONE) każde wywołanie MQCONN jest niezależne od innych. Z tego powodu, gdy ta przykładowa aplikacja tworzy trzecią wartość MQCONN (*CORE), może to raz jeszcze podjąć próbę nawiązania połączenia przez zerwany kanał C, a następnie wybrać jedną z wartości B lub D.

Przykłady wywołań MQCONN
Przykłady użycia komendy MQCONN w celu nawiązania połączenia z konkretnym menedżerem kolejek lub do jednej 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 można użyć wywołania MQCONNX zamiast wywołania MQCONN).

Na komputerach serwera działają dwa menedżery kolejek: jedna o nazwie SALE , a druga o nazwie SALE_BACKUP.



Rysunek 103. Przykład MQCONN

Definicje kanałów w tych przykładach są następujące:

Definicje interfejsu ALE:

```
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 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 ilustrują przykłady MQCONN
Przykłady demonstrują użycie wielu menedżerów kolejek jako systemu zapasowego.

Przypuśćmy, że połączenie komunikacyjne z serwerem 1 jest tymczasowo zerwane. Demonstruje się użycie wielu menedżerów kolejek jako systemu zapasowego.

Każdy przykład obejmuje różne wywołania MQCONN i wyjaśnia, co dzieje się w konkretnym przykładzie, stosując następujące reguły:

1. Tabela definicji kanału klienta (CCDT) jest skanowana w kolejności alfabetycznej dla nazwy menedżera kolejek (pole QMNAME) odpowiadającej temu, który jest podany w wywołaniu MQCONN.
2. Jeśli zostanie znaleziony zgodny element, zostanie użyta definicja kanału.
3. Podejmowana jest próba uruchomienia kanału na komputerze identyfikowany przez nazwę połączenia (CONNAME). Jeśli operacja zakończy się pomyślnie, aplikacja będzie kontynuowana. Wymaga:
   - Program nasłuchujący, który ma być uruchomiony na serwerze.
   - Obiekt nasłuchiwania, który ma być połączony z tym samym menedżerem kolejek, co klient, z którym ma zostać nawiązane połączenie (o ile zostało określone).
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 będzie wyszukiwany w celu uzyskania kolejnego dopasowania. Jeśli dopasowanie zostanie znalezione, przetwarzanie będzie kontynuowane w kroku 1.
5. Jeśli nie zostanie znaleziony żaden zgodny element lub nie ma więcej wpisów w tabeli definicji kanału klienta, a uruchomienie kanału nie powiodło się, aplikacja nie będzie mogła nawiązać połączenia. W wywołaniu MQCONN zwracany jest odpowiedni kod przyczyny i kod zakończenia. Aplikacja może podjąć działanie w oparciu o zwrócone kody przyczyny i zakończenia.

Przykład 1. Nazwa menedżera kolejek zawiera gwiazdkę (*)
W tym przykładzie aplikacja nie dotyczy menedżera kolejek, z którym łączy się dany menedżer kolejek. Aplikacja wysyła wywołanie MQCONN dla nazwy menedżera kolejek, w tym gwiazdkę. Wybrany jest odpowiedni kanał.

Problemy z aplikacją:

```
MQCONN (*SALE)
```

Postępując zgodnie z zasadami, dzieje się tak w tym przypadku:

1. Tabela definicji kanału klienta (CCDT) jest skanowana pod kątem nazwy menedżera kolejek SALE, która jest zgodna z wywołaniem MQCONN aplikacji.
2. Znaleziono definicje kanałów dla produktów ALPHA i BETA .
3. Jeśli jeden kanał ma wartość CLNTWGHT równą 0, ten kanał jest wybierany. Jeśli obie wartości mają wartość CLNTWGHT równą 0, wybierany jest kanał ALPHA , ponieważ jest on pierwszy w kolejności alfabetycznej. Jeśli oba kanały mają niezerową wartość CLNTWGHT, jeden kanał jest wybierany losowo, w oparciu o jego wagę.
4. Podjęto próbę uruchomienia kanału.
5. Jeśli wybrano kanał BETA , próba uruchomienia tego kanału powiodła się.
6. Jeśli wybrano kanał ALPHA , próba uruchomienia go NIE powiedzie się, ponieważ łącze komunikacyjne jest zerwane. Następnie należy wykonać następujące kroki:
   a. Jedynym innym kanałem dla nazwy menedżera kolejek SALE jest BETA.
   b. Podejmowana jest próba uruchomienia tego kanału-operacja ta powiodła się.
7. Sprawdzenie, czy nasłuchiwanie jest uruchomione, pokazuje, że istnieje jeden uruchomiony program nasłuchujący. Nie jest on połączony z menedżerem kolejek produktu SALE , ale dlatego, że parametr wywołania MQI zawiera gwiazdkę (*), 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 łączyć się z określonym menedżerem kolejek. Aplikacja wysyła wywołanie MQCONN dla tej nazwy menedżera kolejek. Wybrany jest odpowiedni kanał.

Aplikacja wymaga połączenia z określonym menedżerem kolejek o nazwie SALE, jak widać w wywołaniu MQI:

```
MQCONN (SALE)
```

Postępując zgodnie z zasadami, dzieje się tak w tym przypadku:

1. Tabela definicji kanału klienta (CCDT) jest skanowana w kolejności alfabetycznej, dla nazwy menedżera kolejek SALE, która jest zgodna z wywołaniem MQCONN aplikacji.
2. Pierwsza znaleziona definicja kanału to ALPHA.
3. Podjęto próbę uruchomienia kanału-operacja ta nie powiodła się, ponieważ łącze komunikacyjne jest zerwane.
4. Tabela definicji kanału klienta zostanie ponownie zeskanowana dla nazwy menedżera kolejek SALE , a nazwa kanału BETA zostanie znaleziona.
5. Podjęto próbę uruchomienia kanału-operacja ta powiodła się.
6. Sprawdzenie, czy nasłuchiwanie jest uruchomione, wskazuje, że jest uruchomiony, ale nie jest on połączony z menedżerem kolejek produktu SALE .
7. W tabeli definicji kanału klienta nie ma kolejnych wpisów. Aplikacja nie może kontynuować działania i otrzymuje kod powrotu MQRC_Q_MGR_NOT_AVAILABLE.

Przykład 3. Nazwa menedżera kolejek jest pusta lub gwiazdka (*)
W tym przykładzie aplikacja nie dotyczy menedżera kolejek, z którym łączy się dany menedżer kolejek. Aplikacja wysyła komendę MQCONN, określając pustą nazwę menedżera kolejek lub gwiazdkę. Wybrany jest odpowiedni kanał.

Jest on traktowany w taki sam sposób, jak produkt “Przykład 1. Nazwa menedżera kolejek zawiera gwiazdkę (*)” na stronie 1026.

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 jest to sytuacja, w której jest uruchamiana z poziomu środowiska klienta. Dostęp do menedżera kolejek jest powiązany z programem nasłuchującym, z którym łączy się kanał.

Problemy z aplikacją:

```
MQCONN ("")
```

lub wersji

```
MQCONN (*)
```

Postępując zgodnie z zasadami, dzieje się tak w tym przypadku:

1. Tabela definicji kanału klienta (CCDT) jest skanowana w kolejności alfabetycznej, dla nazwy menedżera kolejek, która jest pusta i jest zgodna z wywołaniem wywołania MQCONN aplikacji.
2. Pozycja dla kanału o nazwie ALPHA ma nazwę menedżera kolejek w definicji SALE. Wartość ta nie jest zgodna z parametrem wywołania MQCONN, który wymaga, aby nazwa menedżera kolejek była pusta.
3. Następny wpis dotyczy nazwy kanału BETA.
4. `queue manager name` w definicji to SALE. Po raz kolejny wartość ta nie jest zgodna z parametrem wywołania MQCONN, który wymaga, aby nazwa menedżera kolejek była pusta.
5. W tabeli definicji kanału klienta nie ma kolejnych wpisów. Aplikacja nie może kontynuować działania i otrzymuje kod powrotu MQRC_Q_MGR_NOT_AVAILABLE.

Wyzwalanie w środowisku klienta

Komunikaty wysyłane przez aplikacje produktu IBM MQ działające na serwerze IBM MQ MQI clients przyczyniają się do wyzwalania dokładnie w taki sam sposób, jak wszystkie inne komunikaty i mogą być używane do wyzwalania programów zarówno na serwerze, jak i na kliencie.

Wyzwalanie jest szczegółowo wyjaśnione w sekcji Kanały wyzwalane.

Monitor wyzwalacza i aplikacja, która ma być uruchomiona, muszą znajdować się w tym samym systemie.

Domyślne parametry kolejki wyzwalanej są takie same jak w środowisku serwera. W szczególności, jeśli w aplikacji klienckiej nie są określone opcje sterujące punktu synchronizacji MQPMO, umieszczanie komunikatów w kolejce wyzwalanej, która jest lokalna względem menedżera kolejek produktu z/OS , komunikaty są umieszczane w obrębie jednostki pracy. Jeśli warunek wyzwalający zostanie spełniony, komunikat wyzwalacza jest umieszczany w kolejce inicjujący w tej samej jednostce pracy i nie może zostać pobrany przez monitor wyzwalacza do momentu zakończenia jednostki pracy. Proces, który ma zostać wyzwolony, nie zostanie uruchomiony, dopóki jednostka pracy nie zostanie zakończona.

Definicja procesu
Należy zdefiniować definicję procesu na serwerze, ponieważ jest ona powiązana z kolejką, w której ustawiono wyzwalanie.

Obiekt procesu definiuje, co ma zostać wyzwolone. Jeśli klient i serwer nie są uruchomione na tej samej platformie, wszystkie procesy uruchomione przez monitor wyzwalacza muszą definiować *ApplType*, w przeciwnym razie serwer przyjmuje jego definicje domyślne (czyli typ aplikacji, która jest zwykle powiązana z maszyną serwera) i powoduje niepowodzenie.

Jeśli na przykład monitor wyzwalacza jest uruchomiony na kliencie Windows 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, a proces nie powiedzie się.

monitor wyzwalacza
Monitor wyzwalacza udostępniany przez produkty inne niżz/OS IBM MQ działa w środowiskach klienckich dla systemów IBM i IBM i, UNIX, Linux, and Windows .

Aby uruchomić monitor wyzwalacza, wydaj jedną z następujących komend:

- IBM i W systemie IBM i:

```
CALL PGM(QMQM/RUNMQTMC) PARM('-m' QmgrName '-q' InitQ)
```

- ULW Na platformach Windowsplatformy UNIX and Linux :

```
runmqtmc [-m QMgrName] [-q InitQ]
```

Domyślną kolejką inicjujową jest SYSTEM.DEFAULT.INITIATION.QUEUE w domyślnym menedżerze kolejek. Kolejka inicjujący jest w miejscu, w którym monitor wyzwalacza wyszukuje komunikaty wyzwalacza. Następnie wywołuje on programy dla odpowiednich komunikatów wyzwalacza. Ten monitor wyzwalacza obsługuje domyślny typ aplikacji i jest taki sam, jak produkt `runmqtrm` , z wyjątkiem tego, że łączy biblioteki klienckie.

Łańcuch komendy, zbudowany przez monitor wyzwalacza, jest następujący:

1. *ApplicId* z odpowiedniej definicji procesu. *ApplicId* to nazwa programu, który ma zostać uruchomiony, ponieważ zostanie on wprowadzony w wierszu komend.
2. Struktura MQTMC2 ujęta w cudzysłów, która została uzyskana z kolejki inicjuj. Łańcuch komendy jest uruchamiany, który ma ten łańcuch, dokładnie tak, jak jest to podany, w cudzysłowie, aby komenda systemowa zaakceptowała ją jako jeden parametr.
3. *EnvrData* z odpowiedniej definicji procesu.

Monitor wyzwalacza nie widzi, czy istnieje inny komunikat w kolejce inicjujący do momentu zakończenia aplikacji, która została uruchomiona. Jeśli aplikacja ma do wykonania wiele operacji, monitor wyzwalacza może nie nadążać za liczbą przybywających komunikatów wyzwalacza. Istnieją dwa sposoby radzenia sobie z tą sytuacją:

1. Czy uruchomiono więcej monitorów wyzwalaczy

   W przypadku wybrania większej liczby monitorów wyzwalacza można sterować maksymalną liczbą aplikacji, które mogą być uruchamiane w dowolnym momencie.

2. Uruchom uruchomione aplikacje w tle

   Jeśli aplikacje mają być uruchamiane w tle, produkt IBM MQ nie ma żadnych ograniczeń dotyczących liczby aplikacji, które mogą być uruchamiane.

Aby uruchomić uruchomioną aplikację w tle w systemach UNIX and Linux , należy umieścić znak & (ampersand) na końcu *EnvrData* definicji procesu.

Aplikacje CICS (inne niżz/OS)
Program użytkowy innej niżz/OS CICS , który wysyła wywołanie MQCONN lub MQCONNX , musi być zdefiniowany w CEDA jako RESIDENT. Jeśli aplikacja serwera CICS zostanie ponownie dowiązana jako klient, istnieje ryzyko utraty obsługi punktów synchronizacji.

Program użytkowy innej niżz/OS CICS , który wysyła wywołanie MQCONN lub MQCONNX , musi być zdefiniowany w CEDA jako RESIDENT. Aby kod rezydentny był jak najdrobny, można utworzyć dowiązanie do osobnego programu w celu wywołania wywołania MQCONN lub MQCONNX .

Jeśli do zdefiniowania połączenia klienta używana jest zmienna środowiskowa MQSERVER, musi być ona określona w pliku CICSENV.CMD .

Aplikacje produktu IBM MQ mogą być uruchamiane w środowisku serwera IBM MQ lub na kliencie IBM MQ bez zmiany kodu. Jednak w środowisku serwera IBM MQ produkt CICS może działać jako koordynator punktu synchronizacji, a użytkownik korzysta z funkcji EXEC CICS SYNCPOINT i EXEC CICS SYNCPOINT ROLLBACK, a nie **MQCMIT** i **MQBACK**. Jeśli aplikacja CICS jest po prostu relinked as a client, sync point support is lost. Produkty **MQCMIT** i **MQBACK** muszą być używane dla aplikacji działającej na serwerze IBM MQ MQI client.

Windows UNIX Przygotowywanie i uruchamianie aplikacji CICS i Tuxedo

Aby uruchamiać aplikacje produktu CICS i Tuxedo jako aplikacje klienckie, należy użyć różnych bibliotek z tych, które są używane z aplikacjami serwera. Identyfikator użytkownika, pod którym działa aplikacja, również jest inny.

Aby przygotować aplikacje produktu CICS i Tuxedo do uruchamiania jako aplikacje produktu IBM MQ MQI client , należy postępować zgodnie z instrukcjami znajdującymi się w sekcji Konfigurowanie rozszerzonego klienta transakcyjnego.

Należy jednak pamiętać, że informacje, które dotyczą przygotowania aplikacji CICS i Tuxedo, w tym przykładowych programów dostarczanych wraz z produktem IBM MQ, zakłada, że użytkownik przygotowuje aplikacje do działania w systemie serwera IBM MQ . W wyniku tego informacje odnoszą się tylko do bibliotek produktu IBM MQ , które są przeznaczone do użycia w systemie serwera. Przygotowując aplikacje klienckie, należy wykonać następujące czynności:

- Użyj odpowiedniej biblioteki systemu klienta dla powiązań języka, z których korzysta aplikacja. Na przykład:
  - UNIX W przypadku aplikacji napisanych w języku C na serwerze UNIXnależy użyć biblioteki libmqic biblioteki zamiast biblioteki libmqm.
  - Windows W systemach Windows należy użyć biblioteki mqic.lib zamiast katalogu mqm.lib.
- Zamiast bibliotek systemowych serwera wyświetlanych w systemach Tabela 138 na stronie 1030 i Tabela 139 na stronie 1030należy użyć równoważnych bibliotek systemowych klienta. Jeśli biblioteka systemowa serwera nie znajduje się na liście w tych tabelach, należy użyć tej samej biblioteki w systemie klienckim.

Tabela 138. Biblioteki systemowe klienta w systemie UNIX

Biblioteka dla systemu serwera IBM MQ	**Równoważna biblioteka do użycia w systemie klienckim IBM MQ**
libmqmxa	libmqcxa

Tabela 139. 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 pod CICS, zwykle jest on przełączany z użytkownika CICS na identyfikator użytkownika transakcji. Jednak w przypadku uruchamiania aplikacji IBM MQ MQI client w ramach produktu CICSzachowuje uprawnienia uprzywilejowane CICS .

Windows UNIX *Programy przykładowe CICS i Tuxedo*

Programy przykładowe CICS i Tuxedo przeznaczone do użycia w systemach UNIX i Windows .

Tabela 140 na stronie 1031 zawiera przykładowe programy CICS i Tuxedo, które są dostarczane do użytku w systemach klienckich UNIX . Tabela 141 na stronie 1031 zawiera listę równoważnych informacji dla systemów klienckich Windows . Tabele zawierają również listę plików używanych do przygotowania i uruchamiania programów. Opis przykładowych programów można znaleźć w sekcji “Przykład transakcji

CICS” na stronie 1189 i “Korzystanie z przykładów TUXEDO w systemach UNIX i Windows” na stronie 1233.

Tabela 140. Przykładowe programy dla systemów klienckich UNIX		
Opis	**Źródło**	**Moduł wykonywalny**
CICS program	amqscic0.ccs	amqscic
Plik nagłówkowy dla programu CICS	amqscih0.h	-
Program kliencki Tuxedo do umieszczania komunikatów	amqstxpx.c	-
Program kliencki Tuxedo do pobierania wiadomości	amqstxgx.c	-
Program serwerowy 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 141. Przykładowe programy dla systemów klienckich Windows		
Opis	**Źródło**	**Moduł wykonywalny**
CICSTransakcja	amqscic0.ccs	amqscic
Plik nagłówkowy dla transakcji CICS	amqscih0.h	-
Program kliencki Tuxedo do umieszczania komunikatów	amqstxpx.c	-
Program kliencki Tuxedo do pobierania wiadomości	amqstxgx.c	-
Program serwerowy 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	-

Windows UNIX *Komunikat o błędzie AMQ5203w wersji zmodyfikowanej dla aplikacji CICS i Tuxedo*

W przypadku uruchamiania aplikacji CICS lub Tuxedo, które korzystają z rozszerzonego klienta transakcyjnego, mogą być wyświetlane standardowe komunikaty diagnostyczne. Jeden z nich został zmodyfikowany do użycia z rozszerzonym klientem transakcyjnym

Komunikaty, które mogą być wyświetlane w plikach dziennika błędów produktu IBM MQ , są opisane w sekcji Komunikaty diagnostyczne: AMQ4000-9999. Komunikat AMQ5203 został zmodyfikowany do użycia z rozszerzonym klientem transakcyjnym. Poniżej znajduje się tekst zmodyfikowanego komunikatu:

AMQ5203: Wystąpił błąd podczas wywoływania interfejsu XA.

Wyjaśnienie

Numer błędu to & 2, gdzie wartość 1 wskazuje, że podana wartość flag dla & 1 była niepoprawna, 2 wskazuje, że wystąpiła próba użycia bibliotek wielowątkowych i niewielowątkowych w tym samym procesie, 3 wskazuje, że wystąpił błąd z podaną nazwą menedżera kolejek '& 3', 4 wskazuje, że ID menedżera zasobów & 1 jest niepoprawny, 5 wskazuje na to, 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 w momencie, gdy aplikacja nie jest połączona z menedżerem kolejek, 7 wskazuje, że wywołanie XA zostało wykonane w trakcie innego wywołania, 8 wskazuje, że łańcuch xa_info' & 4 'w wywołaniu xa_open zawierał niepoprawną wartość parametru o nazwie' & 5 ', a 9 wskazuje, że łańcuch xa_info' & 4 'w wywołaniu xa_open nie ma wymaganego parametru, nazwa parametru' & 5 '.

Odpowiedź użytkownika

Popraw błąd i spróbuj ponownie wykonać operację.

Windows Przygotowywanie i uruchamianie aplikacji produktu Microsoft Transaction Server

Aby przygotować aplikację MTS do działania jako aplikację IBM MQ MQI client , należy postępować zgodnie z poniższa instrukcjami, aby uzyskać informacje na temat środowiska.

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

Aby przygotować aplikację MTS do uruchamiania jako aplikację 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 w sekcji “Przygotowywanie programów w języku C w programie Windows” na stronie 1123 , ale należy połączyć komponent z biblioteką mqicxa.lib zamiast z biblioteką mqic.lib.
- Jeśli komponent korzysta z klas języka C++ produktu IBM MQ , należy postępować zgodnie z instrukcjami w sekcji “Budowanie programów C++ w systemie Windows” na stronie 561 , ale należy połączyć ten komponent z biblioteką imqx23vn.lib zamiast imqc23vn.lib.
- Jeśli komponent używa powiązań języka Visual Basic dla interfejsu MQI, należy postępować zgodnie z instrukcjami w “Przygotowywanie programów Visual Basic w programie Windows” na stronie 1126 , ale po zdefiniowaniu projektu Visual Basic należy wpisać `MqType=3` w polu **Warunkowe argumenty kompilacji** .
- Jeśli komponent korzysta z klas automatyzacji produktu IBM MQ dla elementu ActiveX (MQAX), należy zdefiniować zmienną środowiskową GMQ_MQ_LIB o wartości `mqic32xa.dll`.

  Można zdefiniować zmienną środowiskową z poziomu aplikacji lub zdefiniować ją w taki sposób, aby jej zasięg był szeroki. Jednak zdefiniowanie go jako całego systemu może spowodować, że dowolna istniejąca aplikacja MQAX, która nie definiuje zmiennej środowiskowej z aplikacji, zachowuje się niepoprawnie.

Przygotowywanie i uruchamianie aplikacji IBM MQ JMS

Aplikacje programu IBM MQ JMS można uruchamiać w trybie klienta z produktem WebSphere Application Server jako menedżerem transakcji. Mogą zostać wyświetlone pewne komunikaty ostrzegawcze.

Aby przygotować i uruchomić aplikacje produktu IBM MQ JMS w trybie klienta z produktem WebSphere Application Server jako menedżerem transakcji, należy postępować zgodnie z instrukcjami w sekcji “użycieIBM MQ classes for JMS” na stronie 83.

Podczas uruchamiania aplikacji klienckiej IBM MQ JMS mogą być wyświetlane następujące komunikaty ostrzegawcze:

MQJE080

Niewystarczające jednostki licencji-uruchom komendę setmqcap

MQJE081

Plik zawierający informacje o jednostce licencji znajduje się w niepoprawnym formacie-uruchom komendę setmqcap

MQJE082

Nie można znaleźć pliku zawierającego informacje o jednostce licencji-uruchom komendę setmqcap

Procedury zewnętrzne, wyjścia funkcji API i usługi instalowalne produktu IBM MQ

Ten temat zawiera odsyłacze do informacji na temat używania i programowania tych programów.

Aby uzyskać informacje na temat korzystania z wyjść użytkownika, wyjść funkcji API i usług instalowalnych w celu rozszerzenia infrastruktury menedżera kolejek, należy zapoznać się z Rozszerzanie obiektów menedżera kolejek.

Informacje na temat pisania i kompilowania wyjść oraz instalowalnych usług można znaleźć w podtematach.

Pojęcia pokrewne

Programy obsługi wyjścia kanału dla kanałów MQI

Odsyłacze pokrewne

Odwołanie do wyjścia funkcji API

Informacje uzupełniające o interfejsie usług instalowalnych

IBM i Informacje uzupełniające o instalowalnym interfejsie usług w systemie IBM i

ULW Zapisywanie wyjść i instalowalnych usług w systemach UNIX, Linux i Windows

Istnieje możliwość pisania i kompilowania wyjść bez łączenia się z żadną biblioteką IBM MQ w systemach UNIX, Linux i Windows.

O tym zadaniu

Ten temat dotyczy tylko systemów UNIX, Linux, and Windows . Szczegółowe informacje na temat pisania wyjść i instalowalnych usług dla innych platform można znaleźć w odpowiednich tematach dotyczących poszczególnych platform.

Jeśli produkt IBM MQ jest zainstalowany w położeniu innym niż domyślne, należy napisać i skompilować wyjścia bez łączenia się z żadną biblioteką produktu IBM MQ .

Istnieje możliwość tworzenia i kompilowania wyjść w systemach UNIX, Linux, and Windows bez łączenia żadnej z tych bibliotek produktu IBM MQ :

- mqmzf
- MQM
- mqmvx
- mqmvxd
- mqic
- mqutl

Istniejące wyjścia, które są powiązane z tymi bibliotekami, nadal działają, pod warunkiem, że w systemach UNIX and Linux IBM MQ jest zainstalowane w domyślnym położeniu.

Procedura

1. Dołącz plik nagłówkowy cmqec.h .

   Dołączenie tego pliku nagłówkowego powoduje automatyczne dołączanie plików nagłówkowych cmqc.h, cmqxc.h i cmqzc.h .
2. Zapisz wyjście tak, aby wywołania MQI i DCI były wykonywane za pomocą struktury MQIEP. Więcej informacji na temat struktury MQIEP można znaleźć w sekcji Struktura MQIEP.
   - Usługi instalowalne
     - Aby wskazać wywołanie MQZEP, należy użyć parametru **Hconfig** .

  - Przed użyciem parametru **Hconfig** należy sprawdzić, czy pierwsze 4 bajty produktu **Hconfig** są zgodne z **StrucId** struktury MQIEP.
  - Więcej informacji na temat pisania instalowalnych komponentów usług znajduje się w sekcji MQIEP.
- Wyjścia funkcji API
  - Aby wskazać wywołanie MQXEP, należy użyć parametru **Hconfig** .
  - Przed użyciem parametru **Hconfig** należy sprawdzić, czy pierwsze 4 bajty produktu **Hconfig** są zgodne z **StrucId** struktury MQIEP.
  - Więcej informacji na temat pisania wyjść funkcji API zawiera sekcja “Pisanie wyjść funkcji API” na stronie 1051.
- Wyjścia kanału
  - Należy użyć parametru **pEntryPoints** struktury MQCXP, aby wskazać wywołania MQI i DCI.
  - Przed użyciem produktu **pEntryPoints**należy sprawdzić, czy numer wersji produktu MQCXP jest w wersji 8 lub nowszej.
  - Więcej informacji na temat pisania wyjść kanału zawiera sekcja “Pisanie programów obsługi wyjścia kanału” na stronie 1062.
- Wyjścia konwersji danych
  - Należy użyć parametru **pEntryPoints** struktury MQDXP, aby wskazać wywołania MQI i DCI.
  - Przed użyciem produktu **pEntryPoints**należy sprawdzić, czy numer wersji produktu MQDXP jest w wersji 2 lub nowszej.
  - Do utworzenia kodu konwersji danych, który korzysta z parametru **pEntryPoints** , można użyć komendy **crtmqcvx** oraz pliku źródłowego amqsvfc0.c . Patrz “Zapisywanie wyjścia konwersji danych dla IBM MQ for Windows” na stronie 1090 oraz “Zapisywanie wyjścia konwersji danych dla systemu IBM MQ w systemach UNIX and Linux” na stronie 1087.
  - Jeśli istnieją wyjścia konwersji danych, które zostały wygenerowane za pomocą komendy **crtmqcvx** , należy ponownie wygenerować wyjście przy użyciu zaktualizowanej komendy.
  - Więcej informacji na temat pisania wyjść konwersji danych zawiera sekcja “Pisanie wyjść konwersji danych” na stronie 1082.
- Wyjścia przed podłączniem
  - Należy użyć parametru **pEntryPoints** struktury MQNXP, aby wskazać wywołania MQI i DCI.
  - Przed użyciem produktu **pEntryPoints**należy sprawdzić, czy numer wersji produktu MQNXP jest w wersji 2 lub nowszej.
  - Więcej informacji na temat pisania wyjść przed podłączaniem znajduje się w sekcji “Odwołanie do definicji połączenia przy użyciu wyjścia wstępnego z połączeniem z repozytorium” na stronie 1092.
- Publikuj wyjścia
  - Aby wskazać wywołania MQI i DCI, należy użyć parametru **pEntryPoints** struktury MQPSXP.
  - Przed użyciem produktu **pEntryPoints**należy sprawdzić, czy numer wersji produktu MQPSXP jest w wersji 2 lub nowszej.
  - Więcej informacji na temat pisania wyjść publikowania znajduje się w sekcji “Pisanie i kompilowanie wyjść publikowania” na stronie 1094.
- Wyjścia obciążenia klastra
  - Aby wskazać wywołania MQXCLWLN, należy użyć parametru **pEntryPoints** struktury MQWXP.
  - Przed użyciem produktu **pEntryPoints**należy sprawdzić, czy numer wersji produktu MQWXP jest w wersji 4 lub nowszej.
  - Więcej informacji na temat pisania wyjść obciążenia klastra zawiera sekcja “Pisanie i kompilowanie wyjść obciążenia klastra” na stronie 1096.

Na przykład w przypadku wyjścia kanału wywołującej wywołanie MQPUT:

```
pChannelExitParms -> pEntryPoints -> MQPUT_Call(pChannelExitParms -> Hconn,
                                                Hobj,
                                                &md,
                                                &pmo,
                                                messlen,
                                                buffer,
                                                &CompCode,
                                                &Reason);
```

Dodatkowe przykłady można znaleźć w publikacji “Korzystanie z przykładowych programów proceduralnych produktu IBM MQ” na stronie 1165.

3. Skompiluj wyjście:
   - Nie należy łączyć się z bibliotekami produktu IBM MQ .
   - Nie należy dołączać wbudowanej ścieżki RPath do żadnych bibliotek produktu IBM MQ w wyjściu.
   - Więcej informacji na temat kompilowania programu obsługi wyjścia można znaleźć w jednym z następujących tematów:
     - Wyjścia funkcji API: “Kompilowanie wyjść funkcji API” na stronie 1053.
     - Wyjścia kanału, wyjścia publikowania, wyjścia obciążenia klastra: “Kompilowanie programów obsługi wyjścia kanału w systemach Windowsi UNIX and Linux” na stronie 1081.
     - Wyjścia konwersji danych: “Pisanie wyjść konwersji danych” na stronie 1082.
4. Umieść wyjście w jednym z następujących miejsc:
   - Ścieżka wyboru, którą można w pełni kwalifikować 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 wyjść 32-bitowych, a *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 **Exits** .
     d. W domyślnym polu ścieżki wyjścia podaj nazwę ścieżki do katalogu, w którym znajduje się program obsługi wyjścia.

   Jeśli wyjście zostanie umieszczone zarówno w określonym katalogu instalacyjnym, jak i w domyślnym katalogu ścieżki, to instalacja programu IBM MQ o nazwie określonej w ścieżce jest używana przez program obsługi wyjścia dla konkretnego katalogu instalacyjnego. Na przykład wyjście jest umieszczane w `/exits/installation2` i w `/exits`, ale nie w `/exits/installation1`. Instalacja produktu IBM MQ `installation2` korzysta z wyjścia z programu `/exits/installation2`. Podczas instalacji IBM MQ `installation1` korzysta z wyjścia z katalogu `/exits` .
5. Jeśli to konieczne, skonfiguruj wyjście:
   - Instalowalne usługi: “Konfigurowanie usług i komponentów” na stronie 1044.
   - Wyjścia funkcji API: “Konfigurowanie wyjść funkcji API” na stronie 1056.
   - Wyjścia kanału: “Konfigurowanie wyjść kanału” na stronie 1082.
   - Publikowanie wyjść: “Konfigurowanie wyjść publikowania” na stronie 1095.
   - Wyjścia z wyprzedzeniem: PreConnect w sekcji pliku konfiguracyjnego klienta.

ULW *Wyjścia funkcji API, które nie są powiązane z biblioteką MQI*

W pewnych okolicznościach należy połączyć istniejące wyjście funkcji API, które nie może zostać ponownie zakodowane w celu użycia wskaźników funkcji MQIEP, z biblioteką API IBM MQ .

Jest to konieczne, dzięki czemu istniejące wyjście funkcji API może zostać pomyślnie załadowane przez konsolidator środowiska wykonawczego systemu do programów, które nie mają już załadowanych wskaźników funkcji.

Uwaga: Te informacje są ograniczone do istniejących wyjść funkcji API, które wywołują wywołania MQI bezpośrednio. Oznacza to, że wyjścia, które nie są używane, MQIEP. Jeśli to możliwe, należy zaplanować ponowne kodowanie wyjścia w celu użycia zamiast nich punktów wejścia MQIEP.

W produkcie IBM MQ 8.0jest to **runmqsc** programu, który nie łączy się bezpośrednio z biblioteką MQI.

Z tego powodu wyjście funkcji API, które nie zostało powiązane z wymaganą biblioteką API produktu IBM MQ lub zakodowane ponownie w celu użycia programu MQIEP, nie może zostać załadowane do produktu **runmqsc**.

Błędy są wyświetlane w dzienniku błędów menedżera kolejek, na przykład AMQ6175: `System nie może dynamicznie załadować biblioteki współużytkowanej`wraz z tekstem kwalifikującym, takim jak `undefined symbol: MQCONN`.

i AMQ7214: `Nie można załadować modułu dla wyjścia funkcji API 'myexitname'`.

Zadania pokrewne

“Zapisywanie wyjść i instalowalnych usług w systemach UNIX, Linux i Windows” na stronie 1033
Istnieje możliwość pisania i kompilowania wyjść bez łączenia się z żadną biblioteką IBM MQ w systemach UNIX, Linux i Windows.

ULW Instalowalne usługi i komponenty dla produktów UNIX, Linux i Windows

Ta sekcja zawiera wprowadzenie do instalowalnych usług oraz powiązanych z nimi funkcji i komponentów. Interfejs do tych funkcji jest udokumentowany w taki sposób, że użytkownik lub dostawcy oprogramowania mogą dostarczać komponenty.

Główne przyczyny udostępniania usług instalowalnych produktu IBM MQ to:

- Aby zapewnić elastyczność wyboru, czy używać komponentów udostępnianych przez produkty IBM MQ , czy też zastąpić je innymi komponentami, czy też je rozszerzyć.
- Aby umożliwić dostawcom udział, udostępniając komponenty, które mogą korzystać z nowych technologii, bez wprowadzania zmian wewnętrznych w produktach IBM MQ .
- Aby umożliwić produktowi IBM MQ korzystanie z nowych technologii w sposób szybszy i tańszy, a także dostarczać produkty wcześniej i po niższych cenach.

Instalowane usługi 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 część centralna wymaga wielu funkcji serwisowych, nazywanych *usługami instalowalnymi*, aby móc wykonywać swoją pracę. Możliwe do zainstalowania usługi to:

- Usługa autoryzacji
- usługa nazw

Każda możliwa do zainstalowania usługa jest pokrewny zestawem funkcji zaimplementowanych przy użyciu co najmniej jednego *komponentów usług*. Każdy komponent jest wywoływany przy użyciu poprawnie zaprojektowanego, publicznie dostępnego interfejsu. Umożliwia to niezależnym dostawcom oprogramowania i innym osobom trzecim udostępnianie instalowalnych komponentów w celu rozszerzenia lub zastąpienia tych, które są dostarczane przez produkty IBM MQ . Tabela 142 na stronie 1037 podsumowuje usługi i komponenty, które mogą być używane.

Tabela 142. Podsumowanie instalowalnych komponentów usług			
usługa instalowalna	**Dostarczony komponent**	**Funkcja**	**Wymagania**
Usługa autoryzacji	menedżer uprawnień obiektu (OAM)	Udostępnia sprawdzanie autoryzacji komend i wywołań MQI. Użytkownicy mogą napisać własny komponent w celu rozszerzenia lub zastąpienia OAM. Na przykład, aby sprawdzić, czy identyfikator użytkownika ma uprawnienia do otwierania kolejki.	(Przyjęto odpowiednie narzędzia autoryzacji platformy)
usługa nazw	Brak	Zapewnia obsługę menedżera kolejek w celu wyszukiwania nazwy menedżera kolejek, do którego należy określona kolejka. • Zdefiniowany przez użytkownika	• Menedżer nazw innej firmy lub napisanej przez użytkownika

Interfejs usług instalowalnych jest opisany w sekcji Informacje uzupełniające dotyczące interfejsu usług instalowalnych.

Zadania pokrewne
Konfigurowanie usług instalowalnych

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 zdefiniowana jest kolejka
- Wstawianie nazwy kolejki do katalogu usługi
- Usuwanie nazwy kolejki z katalogu usługi

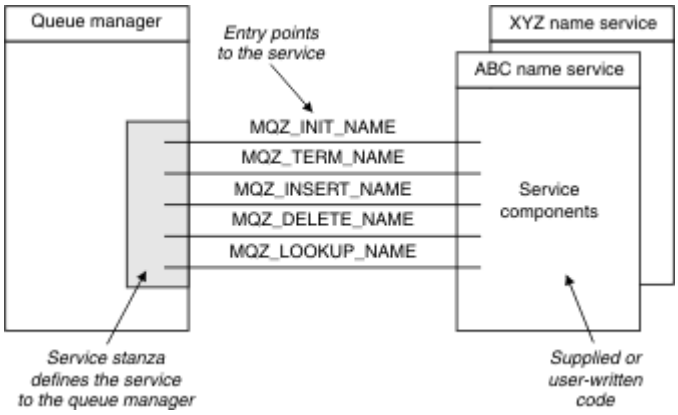
Zawiera on także funkcje inicjowania i zakończenia.

Instalowalna usługa jest udostępniana przez jeden lub więcej komponentów usług. Każdy komponent może wykonywać niektóre lub wszystkie funkcje, które są zdefiniowane dla tej usługi. Na przykład w produkcie IBM MQ for AIXdostarczony komponent usługi autoryzacji, OAM, wykonuje wszystkie dostępne funkcje. Więcej informacji zawiera sekcja “Interfejs usługi autoryzacji” na stronie 1041. Komponent jest również odpowiedzialny za zarządzanie dowolnymi zasobami lub oprogramowaniem (na przykład katalogiem LDAP), które musi implementować usługę. Pliki konfiguracyjne udostępniają standardowy sposób ładowania komponentu i określania adresów podprogramów funkcjonalnych, które udostępnia.

Rysunek 104 na stronie 1038 pokazuje, w jaki sposób usługi i komponenty są powiązane:

- Usługa jest zdefiniowana w menedżerze kolejek przez sekcje w pliku konfiguracyjnym.
- Każda usługa jest obsługiwana przez podany kod w menedżerze kolejek. Użytkownicy nie mogą zmieniać tego kodu i w związku z tym nie mogą tworzyć własnych usług.

- Każda usługa jest implementowana przez jeden lub kilka komponentów. Te komponenty mogą być dostarczane razem z produktem lub użytkownikami. Można wywołać wiele komponentów dla usługi, z których każda obsługuje różne urządzenia w ramach usługi.
- Punkty wejścia łączą komponenty usługi z kodem pomocowym w menedżerze kolejek.



Rysunek 104. Zrozumienie usług, komponentów i punktów wejścia

Punkty wejścia

Każdy komponent usługi jest reprezentowany przez listę adresów punktów wejścia procedur, które obsługują określoną usługę instalowalną. Usługa instalowalna definiuje funkcję, która ma być wykonywana przez każdą procedurę.

Porządkowanie komponentów usługi po ich skonfigurowaniu definiuje kolejność, w jakiej punkty wejścia są wywoływane w próbie spełnienia żądania dla usługi.

W dostarczonym pliku nagłówkowego `cmqzc.h`podane punkty wejścia dla każdej usługi mają przedrostek MQZID_.

Jeśli usługi są obecne, usługi są ładowane w predefiniowanym porządku. Poniższa lista przedstawia usługi oraz kolejność, w jakiej zostały zainicjowane.

1. `NameService`
2. `AuthorizationService`
3. `UserIdentifierService`

`AuthorizationService` jest jedyną usługą, która jest skonfigurowana domyślnie. Skonfiguruj `NameService` i `UserIdentifierService` ręcznie, jeśli mają być używane.

Usługi i komponenty usług mają odwzorowanie jeden-do-jednego lub jeden do wielu. Dla każdej usługi można zdefiniować wiele komponentów usług. W systemach UNIX and Linux wartość Service w sekcji `ServiceComponent` musi być zgodna z wartością nazwy 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 *nazwa_menedżera_kolejek* to nazwa menedżera kolejek.

W przypadku systemów UNIX and Linux komponenty usług są uruchamiane w kolejności, w jakiej są one zdefiniowane w pliku qm.ini . W systemie Windows, ponieważ używany jest rejestr Windows , produkt IBM MQ wysyła wywołanie **RegEnumKey** , które zwraca wartości w kolejności alfabetycznej. Dlatego też w systemie Windows usługi są wywoływane w kolejności alfabetycznej, ponieważ są one zdefiniowane w rejestrze.

Kolejność definicji `ServiceComponent` jest istotna. Ta kolejność narzuca kolejność, w jakiej komponenty są uruchamiane 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 przesłonięcia domyślnego OAM. Jeśli

nie określono wartości MQCACF_SERVICE_COMPONENT , do przetwarzania żądania używany jest pierwszy komponent napotkany w kolejności alfabetycznej, a nazwa tego komponentu jest używana.

Kody powrotu

Komponenty usług udostępniają menedżerowi kolejek kody powrotu do raportowania w różnych warunkach. Informują one o powodzeniu lub niepowodzeniu operacji, a także wskazują, czy menedżer kolejek ma przejść do następnego komponentu usługi. Ten wskaźnik zawiera osobny parametr *Continuation* .

Dane komponentu

Pojedynczy komponent usługi może wymagać, aby dane były współużytkowane przez różne funkcje. Usługi instalowalne udostępniają opcjonalny obszar danych, który ma być przekazywany w każdym wywołaniu komponentu usługi. Ten obszar danych jest przeznaczony do wyłącznego korzystania z komponentu usługi. Jest on współużytkowany przez wszystkie wywołania danej funkcji, nawet jeśli są one wykonywane z różnych przestrzeni adresowych lub procesów. Gwarantowany jest adresowalny z komponentu usługi za każdym razem, gdy jest wywoływany. Wielkość tego obszaru musi być zadeklarowane w sekcji *ServiceComponent* .

Inicjowanie i kończenia komponentów
Użycie opcji inicjowania i zakończenia komponentu.

Gdy wywoływana jest procedura inicjowania komponentu, musi ona wywołać funkcję menedżera kolejek **MQZEP** 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 przy użyciu podstawowej opcji inicjowania, zanim zostanie wywołany w inny sposób.

Komponent może być wywoływany z dodatkową opcją inicjowania na niektórych platformach. Na przykład może być wywoływany jeden raz dla każdego procesu systemu operacyjnego, wątku lub czynności, do której dostęp jest uzyskiwany przez usługę.

Jeśli używana jest inicjalizacja drugorzędna:

- Komponent może być wywoływany więcej niż jeden raz dla inicjowania dodatkowego. W przypadku każdego takiego wywołania, gdy usługa nie jest już potrzebna, zostanie wysłane zgodne wywołanie dla zakończenia drugorzędnego.

  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 (wywołując MQZEP) za każdym razem, gdy wywoływany jest komponent dla inicjowania podstawowego i dodatkowego.
- Dla komponentu jest używana tylko jedna kopia danych komponentu. Dla każdego inicjowania dodatkowego nie ma innej kopii.
- Komponent nie jest wywoływany w przypadku innych wywołań usługi (z procesu systemu operacyjnego, wątku lub zadania, w zależności od przypadku) przed przeprowadzonym wtórnym zainicjowaniem.
- Komponent musi ustawić parametr **Version** na tę samą wartość dla inicjowania podstawowego i dodatkowego.

Komponent jest zawsze wywoływany po raz pierwszy z podstawową opcją kończenia, gdy nie jest już wymagany. Do tego komponentu nie są wykonywane żadne dodatkowe wywołania.

Komponent jest wywoływany z opcją zakończenia drugorzędnego, jeśli został wywołany w celu zainicjowania dodatkowego.

menedżer uprawnień obiektu (OAM)
Komponent usługi autoryzacji dostarczany razem z produktami IBM MQ jest nazywany menedżerem uprawnień do obiektów (Object Authority Manager-OAM).

Domyślnie program OAM jest aktywny i działa z komendami kontrolnymi **dspmqaut** (uprawnienie do wyświetlania), **dmpmqaut** (uprawnienie do zrzutu) i **setmqaut** (uprawnienie do ustawiania lub resetowania).

Składnia tych komend oraz sposób ich użycia są opisane w sekcji Administrowanie za pomocą komend sterujących.

OAM współpracuje z *jednostką* nazwy użytkownika lub grupy:

- Linux UNIX W systemach UNIX and Linux jednostka główna to identyfikator użytkownika lub identyfikator powiązany z programem użytkowym działającym w imieniu użytkownika; grupa jest zdefiniowaną przez system kolekcją nazw użytkowników.
- Windows W systemach Windows nazwa użytkownika to Windows ID użytkownika lub identyfikator powiązany z programem użytkowym działającym w imieniu użytkownika. Grupa to grupa Windows .

Autoryzacje mogą być nadawane lub odbierane na poziomie głównym lub grupowym.

Po wykonaniu żądania MQI lub wydania komendy OAM sprawdza, czy jednostka powiązana z operacją ma autoryzację do wykonania żądanej operacji i uzyskania dostępu do określonych zasobów menedżera kolejek.

Usługa autoryzacji umożliwia rozszerzanie lub zastępowanie sprawdzanych uprawnień dla menedżerów kolejek poprzez napisanie własnego komponentu usługi autoryzacji.

usługa nazw
Usługa nazw to instalowalna usługa, która udostępnia 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 zdalnych kolejek na potrzeby danych wyjściowych, tak jak w przypadku kolejek lokalnych. Usługa nazw nie jest wywoływana dla obiektów innych niż kolejki.

Uwaga: W kolejkach zdalnych musi być ustawiony atrybut **Scope** (CELL).

Gdy aplikacja otwiera kolejkę, wyszukuje nazwę kolejki najpierw w katalogu menedżera kolejek. Jeśli nie znajdzie go tam, wyszukuje tak wiele usług nazw, które zostały skonfigurowane, dopóki nie znajdzie takiego, który rozpoznaje nazwę kolejki. Jeśli żadna nazwa nie rozpozna nazwy, otwarcie nie powiedzie się.

Usługa nazw zwraca menedżera kolejek, który jest właścicielem dla tej kolejki. Menedżer kolejek kontynuuje działanie z żądaniem MQOPEN tak, jakby komenda określiła nazwę kolejki i menedżera kolejek w oryginalnym żądaniu.

Interfejs usługi nazw (NSI) jest częścią środowiska produktu IBM MQ .

Jak działa usługa nazw

Jeśli definicja kolejki określa atrybut **Scope** jako menedżer kolejek, to znaczy wartość SCOPE (QMGR) w MQSC, definicja kolejki (wraz ze wszystkimi atrybutami kolejki) jest przechowywana tylko w katalogu menedżera kolejek. Nie może to być zastąpione przez usługę instalowalną.

Jeśli definicja kolejki określa atrybut **Scope** jako komórkę, to znaczy wartość SCOPE (CELL) w MQSC, to definicja kolejki jest ponownie zapisywana w katalogu menedżera kolejek wraz ze wszystkimi atrybutami kolejki. Jednak nazwa kolejki i menedżera kolejek są również przechowywane w usłudze nazw. Jeśli żadna usługa nie jest dostępna, która może przechowywać te informacje, nie można zdefiniować kolejki z komórką produktu *Scope* .

Katalog, w którym przechowywane są informacje, może być zarządzany przez usługę lub usługa może korzystać z usługi bazowej, na przykład katalogu LDAP, w tym celu. W obu przypadkach definicje zapisane w katalogu muszą być trwałe, nawet po zakończeniu działania komponentu i menedżera kolejek, dopóki nie zostaną jawnie usunięte.

Uwaga:

1. Aby wysłać komunikat do lokalnej definicji kolejki hosta zdalnego (z zasięgiem komórki) w innym menedżerze kolejek w komórce katalogu nazw, należy zdefiniować kanał.
2. Nie można pobrać komunikatów bezpośrednio z kolejki zdalnej, nawet jeśli ma on zasięg komórki.
3. Podczas wysyłania do kolejki z zasięgiem komórki CELL nie jest wymagana żadna definicja kolejki zdalnej.
4. Usługa nadawania nazw centralnie definiuje kolejkę docelową, mimo że do docelowego menedżera kolejek i pary definicji kanału potrzebna jest kolejka transmisji. 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, to kolejka transmisji w systemie lokalnym musi mieć również nazwę QM01.

Interfejs usługi autoryzacji
Usługa autoryzacji udostępnia punkty wejścia do użycia przez menedżer kolejek.

Punkty wejścia są następujące:

MQZ_AUTHENTICATE_USER
Uwierzytelnia identyfikator użytkownika i hasło oraz może ustawiać pola kontekstu tożsamości.

MQZ_CHECK_AUTHORITY
Sprawdza, czy jednostka ma uprawnienia do wykonywania jednej lub większej liczby operacji na określonym obiekcie.

MQZ_CHECK_PRIVILEGED
Sprawdza, czy określony użytkownik jest użytkownikiem uprzywilejowanym.

MQZ_COPY_ALL_AUTHORITY,
Kopiuje wszystkie bieżące autoryzacje, które istnieją dla obiektu, do którego istnieje odwołanie, do innego obiektu.

MQZ_DELETE_AUTHORITY,
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 przydzielone przydzielone zasoby.

MQZ_GET_AUTHORITY,
Pobiera uprawnienia, które jednostka musi uzyskać, aby uzyskać dostęp do określonego obiektu.

Uprawnienie MQZ_GET_EXPLICIT_AUTHORITY
Uzyskuje uprawnienie do dostępu do określonego obiektu przez nazwaną grupę (ale bez dodatkowego uprawnienia grupy **nobody**) lub uprawnienie, do którego ma dostęp grupa podstawowa nazwanego użytkownika, aby uzyskać dostęp do określonego obiektu.

MQZ_INIT_AUTHORITY,
Inicjuje komponent usługi autoryzacji.

MQZ_INQUIRE
Wysyła zapytania do obsługiwanych funkcji usługi autoryzacji.

MQZ_REFRESH_CACHE
Odśwież wszystkie autoryzacje.

MQZ_SET_AUTHORITY,
Ustawia uprawnienia, które jednostka ma do określonego obiektu.

MQZ_TERM_AUTHORITY,
Przerywa komponent usługi autoryzacji.

Dodatkowo 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 zabezpieczeń serwera Windows (NT SID).

Nazwy te są definiowane jako **typedef** w pliku nagłówkowego `cmqzc.h`, który może być używany do tworzenia prototypów funkcji komponentu.

Funkcja inicjowania (**MQZ_INIT_AUTHORITY**) musi być głównym punktem wejścia dla komponentu. Pozostałe 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
Przerwij komponent usługi nazw.

MQZ_LOOKUP_NAME
Wyszukaj nazwę menedżera kolejek dla podanej kolejki.

MQZ_INSERT_NAME
Wstaw wpis zawierający nazwę menedżera kolejek będącego właścicielem określonej kolejki do katalogu używanego przez usługę.

MQZ_DELETE_NAME
Usuń pozycję dla podanej kolejki z katalogu używanego przez usługę.

Jeśli skonfigurowana jest więcej niż jedna usługa nazw:

- W przypadku wyszukiwania funkcja MQZ_LOOKUP_NAME jest wywoływana dla każdej usługi na liście, dopóki nie zostanie rozstrzygnięta nazwa kolejki (chyba że jakikolwiek 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 istnieje więcej niż jeden komponent, który obsługuje funkcje wstawiania i usuwania. Jednak komponent, który obsługuje wyszukiwanie, jest możliwy do wykonania i może być używany na przykład jako ostatni komponent na liście w celu rozstrzygnięcia dowolnej nazwy, która nie jest znana przez żaden inny komponent 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 prototypowania funkcji serwisowych, aby upewnić się, że parametry są poprawne.

Plik nagłówkowy, który zawiera 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, który został dodany przez funkcję inicjowania, przy użyciu wywołania MQZEP.

Korzystanie z wielu komponentów usług
Dla usługi można zainstalować więcej niż jeden komponent. Umożliwia to komponentom udostępnianie tylko częściowych implementacji usługi, a także oparcie się na innych komponentach w celu udostępnienia pozostałych funkcji.

Przykład użycia wielu komponentów

Załóżmy, że tworzysz dwa komponenty usług nazw o nazwach ABC_name_serv i XYZ_name_serv.

ABC_name_serv
Ten komponent obsługuje wstawienie nazwy lub usunięcie 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 lub usuwania jej z katalogu usług.

Komponent ABC_name_serv przechowuje bazę danych nazw kolejek i korzysta z dwóch prostych algorytmów do wstawiania lub usuwania nazwy z katalogu usług.

Komponent XYZ_name_serv korzysta z prostego algorytmu, który zwraca stałą nazwę menedżera kolejek dla każdej nazwy kolejki, z którą jest wywoływany. Nie posiada on bazy danych nazw kolejek i dlatego nie obsługuje funkcji wstawiania i usuwania.

Komponenty są instalowane w tym samym menedżerze kolejek. Sekcje *ServiceComponent* są zamawiane w taki sposób, że komponent ABC_name_serv jest wywoływany jako pierwszy. Wszystkie wywołania do wstawienia lub usunięcia kolejki w katalogu komponentu są obsługiwane przez komponent ABC_name_serv ; Jest to jedyna, która implementuje te funkcje. Jednak wywołanie wyszukiwania, którego komponent ABC_name_serv nie może rozstrzygnąć, jest przekazywane do komponentu wyszukiwania tylko w komponencie XYZ_name_serv. Ten komponent dostarcza nazwę menedżera kolejek z prostego algorytmu.

Pomijanie punktów wejścia podczas korzystania z wielu komponentów

Jeśli użytkownik zdecyduje się na użycie wielu komponentów w celu udostępnienia usługi, może zaprojektować komponent usługi, który nie implementuje określonych funkcji. Środowisko usług instalowalnych nie zawiera żadnych ograniczeń, które można pominąć. Jednak w przypadku konkretnych usług instalowalnych pominięcie jednej lub większej liczby funkcji może być logicznie niespójne z celem usługi.

Przykład punktów wejścia używanych z wieloma komponentami

Tabela 143 na stronie 1043 przedstawia przykład usługi nazw możliwych do zainstalowania, dla której zostały zainstalowane dwa komponenty. Każdy z nich obsługuje inny zestaw funkcji powiązanych z daną usługą instalowalną. W przypadku funkcji wstawiania, najpierw wywoływana jest punkt wejścia komponentu ABC. Punkty wejścia, które nie zostały zdefiniowane dla usługi (za pomocą komendy **MQZEP**) Przyjmuje się, że ma wartość NULL. Punkt wejścia dla inicjowania znajduje się w tabeli, ale nie jest to wymagane, ponieważ inicjowanie jest wykonywane przez główny punkt wejścia komponentu.

Gdy menedżer kolejek musi korzystać z usługi instalowalnej, używa ona punktów wejścia zdefiniowanych dla tej usługi (kolumny w programie Tabela 143 na stronie 1043). Po włączeniu każdego z komponentów menedżer kolejek określa adres procedury, która implementuje wymaganą funkcję. Następnie wywołuje on procedurę, jeśli istnieje. Jeśli operacja zakończy się pomyślnie, menedżer kolejek używa dowolnych wyników i informacji o statusie.

Tabela 143. Przykład punktów wejścia dla instalowalnej usługi

Numer funkcji	**Komponent usługi nazw ABC**	**Komponent usługi nazw XYZ**
MQZID_INIT_NAME (inicjowanie)	ABC_initialize ()	XYZ_initialize ()
MQZID_TERM_NAME (Przerwij)	ABC_terminate ()	XYZ_terminate ()
MQZID_INSERT_NAME (Wstaw)	ABC_Insert ()	NULL
MQZID_DELETE_NAME (Usunięcie)	ABC_Delete ()	NULL

Tabela 143. Przykład punktów wejścia dla instalowalnej usługi (kontynuacja)		
Numer funkcji	**Komponent usługi nazw ABC**	**Komponent usługi nazw XYZ**
MQZID_LOOKUP_NAME (Wyszukiwanie)	NULL	XYZ_Lookup ()

Jeśli procedura nie istnieje, menedżer kolejek będzie powtarzał ten proces dla następnego komponentu na liście. Dodatkowo, jeśli procedura istnieje, ale zwraca kod wskazujący, że nie może wykonać operacji, próba jest kontynuowana z następnym dostępnym komponentem. Podprogramy w komponentach usług mogą zwracać kod wskazujący, że nie należy wykonywać żadnych dalszych prób wykonania operacji.

Konfigurowanie usług i komponentów

Komponenty usług są konfigurowane przy użyciu 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 w celu zdefiniowania usługi dla menedżera kolejek i określ położenie modułu:
   - Każda używana usługa musi mieć sekcję `Service` , która definiuje usługę dla menedżera kolejek. Więcej informacji na ten temat zawiera sekcja Sekcja usługi w pliku qm.ini.
   - Dla każdego komponentu w ramach usługi musi istnieć sekcja `ServiceComponent` . Ta sekcja identyfikuje nazwę i ścieżkę modułu zawierającego kod dla tego komponentu. Więcej informacji na ten temat zawiera sekcja SekcjaServiceComponent w pliku qm.ini.

   Komponent usługi autoryzacji, znany jako Object Authority Manager (OAM), jest dostarczany razem z produktem. Podczas tworzenia menedżera kolejek plik konfiguracyjny menedżera kolejek (lub rejestr w systemach Windows) jest automatycznie aktualizowany w taki sposób, aby obejmował odpowiednie sekcje dla usługi autoryzacji i dla komponentu domyślnego (OAM). W przypadku pozostałych komponentów należy ręcznie skonfigurować plik konfiguracyjny menedżera kolejek.

   Kod dla każdego komponentu usługi jest ładowany do menedżera kolejek po uruchomieniu menedżera kolejek przy użyciu powiązania dynamicznego, w którym jest to obsługiwane na platformie.
2. Zatrzymaj i zrestartuj menedżer kolejek, aby aktywować komponent.

Odsyłacze pokrewne

Sekcja Service w 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. Aby wykonać tę komendę, wydaj komendę **REFRESH SECURITY** .

Uwaga: W przypadku zmiany autoryzacji za pomocą komendy `setmqaut` , OAM wprowadza takie zmiany natychmiast.

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

Te informacje umożliwiają zapoznanie się z instalowalnymi usługami oraz powiązanymi z nimi funkcjami i komponentami. Interfejs do tych funkcji jest udokumentowany w taki sposób, że użytkownik lub dostawcy oprogramowania mogą dostarczać komponenty.

Główne przyczyny udostępniania usług instalowalnych produktu IBM MQ to:

- Aby zapewnić elastyczność wyboru, czy używać komponentów udostępnianych przez produkt IBM MQ for IBM i, czy też zastąpić je innym, czy też je rozszerzyć.
- Aby umożliwić dostawcom uczestnictwo, udostępniając komponenty, które mogą korzystać z nowych technologii, bez wprowadzania zmian wewnętrznych do produktu IBM MQ for IBM i.
- Aby umożliwić produktowi IBM MQ korzystanie z nowych technologii w sposób szybszy i tańszy, a także dostarczać produkty wcześniej i po niższych cenach.

Instalowane usługi 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 część centralna wymaga wielu funkcji serwisowych, nazywanych *usługami instalowalnymi*, aby móc wykonywać swoją pracę. Instalowalną usługą dostępną w produkcie IBM MQ for IBM i jest usługa autoryzacji.

Każda możliwa do zainstalowania usługa jest pokrewny zestawem funkcji zaimplementowanych przy użyciu co najmniej jednego *komponentów usług*. Każdy komponent jest wywoływany przy użyciu poprawnie zaprojektowanego, publicznie dostępnego interfejsu. Umożliwia to niezależnym dostawcom oprogramowania i innym osobom trzecim udostępnienie instalowalnych komponentów w celu rozszerzenia lub zastąpienia tych dostarczonych przez produkt IBM MQ for IBM i. Tabela 144 na stronie 1045 podsumowuje obsługę usługi autoryzacji.

Tabela 144. Podsumowanie komponentów usługi autoryzacji

Dostarczony komponent	**Funkcja**	**Wymagania**
menedżer uprawnień obiektu (OAM)	Udostępnia sprawdzanie autoryzacji komend i wywołań MQI. Użytkownicy mogą napisać własny komponent w celu rozszerzenia lub zastąpienia OAM.	(Przyjęto odpowiednie narzędzia autoryzacji platformy)
Komponent usługi nazw DCE **Uwaga:** Środowisko DCE jest obsługiwane tylko w wersjach produktu IBM MQ wcześniejszych niż V6.0.	• Umożliwia menedżerom kolejek współużytkowanie kolejek, lub • Zdefiniowany przez użytkownika **Uwaga:** W kolejkach współużytkowanych musi być ustawiony atrybut **Scope** CELL.	• DCE jest wymagane dla dostarczanego komponentu, lub • Menedżer nazw innej firmy lub napisanej przez użytkownika

IBM i *Funkcje i komponenty w systemie IBM i*

Te informacje umożliwiają poznanie funkcji i komponentów, punktów wejścia, kodów powrotu i danych komponentów, których można użyć w produkcie 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 zdefiniowana jest kolejka
- Wstawianie nazwy kolejki do katalogu usługi
- Usuwanie nazwy kolejki z katalogu usługi

Zawiera on także funkcje inicjowania i zakończenia.

Instalowalna usługa jest udostępniana przez jeden lub więcej komponentów usług. Każdy komponent może wykonywać niektóre lub wszystkie funkcje, które są zdefiniowane dla tej usługi. Komponent jest również odpowiedzialny za zarządzanie zasobami bazowymi lub oprogramowaniem, których potrzebuje do zaimplementowania usługi. Pliki konfiguracyjne udostępniają standardowy sposób ładowania komponentu i określania adresów podprogramów funkcjonalnych, które udostępnia.

Usługi i komponenty są powiązane w następujący sposób:

- Usługa jest zdefiniowana w menedżerze kolejek przez sekcje w pliku konfiguracyjnym.

- Każda usługa jest obsługiwana przez podany kod w menedżerze kolejek. Użytkownicy nie mogą zmieniać tego kodu i w związku z tym nie mogą tworzyć własnych usług.
- Każda usługa jest implementowana przez jeden lub kilka komponentów. Te komponenty mogą być dostarczane razem z produktem lub użytkownikami. Można wywołać wiele komponentów dla usługi, z których każda obsługuje różne urządzenia w ramach usługi.
- Punkty wejścia łączą komponenty usługi z kodem pomocowym w menedżerze kolejek.

Punkty wejścia

Każdy komponent usługi jest reprezentowany przez listę adresów punktów wejścia procedur, które obsługują określoną usługę instalowalną. Usługa instalowalna definiuje funkcję, która ma być wykonywana przez każdą procedurę. Porządkowanie komponentów usługi po ich skonfigurowaniu definiuje kolejność, w jakiej punkty wejścia są wywoływane w próbie spełnienia żądania dla usługi. W dostarczonym pliku nagłówkowego `cmqzc.h` podane punkty wejścia dla każdej usługi mają przedrostek MQZID_.

Kody powrotu

Komponenty usług udostępniają menedżerowi kolejek kody powrotu do raportowania w różnych warunkach. Informują one o powodzeniu lub niepowodzeniu operacji, a także wskazują, czy menedżer kolejek ma przejść do następnego komponentu usługi. Ten wskaźnik zawiera osobny parametr *Continuation* .

Dane komponentu

Pojedynczy komponent usługi może wymagać, aby dane były współużytkowane przez różne funkcje. Usługi instalowalne udostępniają opcjonalny obszar danych, który ma być przekazywany przy każdym wywołaniu danego komponentu usługi. Ten obszar danych jest przeznaczony do wyłącznego korzystania z komponentu usługi. Jest on współużytkowany przez wszystkie wywołania danej funkcji, nawet jeśli są one wykonywane z różnych przestrzeni adresowych lub procesów. Gwarantowany jest adresowalny z komponentu usługi za każdym razem, gdy jest wywoływany. Wielkość tego obszaru musi być zadeklarowane w sekcji *`ServiceComponent`* .

IBM i *Inicjowanie w systemie IBM i*

Gdy wywoływana jest procedura inicjowania komponentu, musi ona wywoływać 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.

Podstawowe inicjowanie
Komponent jest zawsze wywoływany z tą opcją jeden raz, zanim zostanie wywołany w jakikolwiek inny sposób.

Inicjowanie wtórne
Komponent może być wywoływany z tą opcją na niektórych platformach. Na przykład może być wywoływany jeden raz dla każdego procesu systemu operacyjnego, wątku lub czynności, do której dostęp jest uzyskiwany przez usługę.

Jeśli używana jest inicjalizacja drugorzędna:

- Komponent może być wywoływany więcej niż jeden raz dla inicjowania dodatkowego. W przypadku każdego takiego wywołania, gdy usługa nie jest już potrzebna, zostanie wysłane zgodne wywołanie dla zakończenia drugorzędnego.

  W przypadku usług autoryzacji jest to wywołanie MQZ_TERM_AUTHORITY.
- Punkty wejścia muszą być ponownie określone (wywołując MQZEP) za każdym razem, gdy wywoływany jest komponent dla inicjowania podstawowego i dodatkowego.
- Dla komponentu jest używana tylko jedna kopia danych komponentu. Dla każdego inicjowania dodatkowego nie ma innej kopii.

- Komponent nie jest wywoływany w przypadku innych wywołań usługi (z procesu systemu operacyjnego, wątku lub zadania, w zależności od przypadku) przed przeprowadzonym wtórnym zainicjowaniem.
- Komponent musi ustawić parametr **Version** na tę samą wartość dla inicjowania podstawowego i dodatkowego.

Zakończenie podstawowe
Komponent jest zawsze uruchamiany z tą opcją raz, gdy nie jest już wymagany. Do tego komponentu nie są wykonywane żadne dodatkowe 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 określ położenie modułu:
   - Każda używana usługa musi mieć sekcję `Service` , która definiuje usługę dla menedżera kolejek. Więcej informacji na ten temat zawiera sekcja Sekcja usługi w pliku qm.ini.
   - Dla każdego komponentu w ramach usługi musi istnieć sekcja `ServiceComponent` . Ta sekcja identyfikuje nazwę i ścieżkę modułu zawierającego kod dla tego komponentu. Więcej informacji na ten temat zawiera sekcja SekcjaServiceComponent w pliku qm.ini.

   Komponent usługi autoryzacji, znany jako Object Authority Manager (OAM), jest dostarczany razem z produktem. Podczas tworzenia menedżera kolejek plik konfiguracyjny menedżera kolejek jest automatycznie aktualizowany w taki sposób, aby zawierał odpowiednie sekcje dla usługi autoryzacji i dla komponentu domyślnego (OAM). W przypadku pozostałych komponentów należy ręcznie skonfigurować plik konfiguracyjny menedżera kolejek.

   Kod dla każdego komponentu usługi jest ładowany do menedżera kolejek po uruchomieniu menedżera kolejek przy użyciu powiązania dynamicznego, w którym jest to obsługiwane na platformie.
2.

IBM i *Tworzenie własnego komponentu usługi w systemie IBM i*

Informacje zawarte w tej sekcji umożliwiają poznanie sposobu tworzenia komponentu usługi w produkcie 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 ze współużytkowanymi bibliotekami `libmqm*` i `libmqmzf*`.

  Uwaga: Ponieważ agent może być uruchamiany w środowisku wielowątkowym, należy zbudować system OAM, aby był uruchamiany w środowisku wielowątkowym. Obejmuje to używanie wersji z wątkami `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 to instalowalna usługa, która umożliwia menedżerom kolejek wywoływanie obiektów autoryzacji, na przykład sprawdzanie, czy ID użytkownika ma uprawnienia do otwierania kolejki.

Ta usługa jest komponentem interfejsu SEI (Security włączeniu interfejsu IBM MQ), który jest częścią środowiska produktu IBM MQ . Omówione zostały następujące tematy:

- “menedżer uprawnień obiektu (OAM)” na stronie 1048
- “Definiowanie usługi w systemie operacyjnym” na stronie 1048
- “Konfigurowanie sekcji usług autoryzacji” na stronie 1048
- “Interfejs usługi autoryzacji w systemie IBM i” na stronie 1049

menedżer uprawnień obiektu (OAM)

Komponent usługi autoryzacji dostarczany z produktami IBM MQ jest nazywany menedżerem uprawnień do obiektów (Object Authority Manager-OAM). Domyślnie OAM jest aktywny i działa z następującymi komendami:

- Praca z uprawnieniami **WRKMQMAUT**
- **WRKMQMAUTD** praca z danymi uprawnień
- Uprawnienie do wyświetlania obiektu **DSPMQMAUT**
- **GRTMQMAUT** nadawanie uprawnień do obiektu
- **RVKMQMAUT** unieważnienie uprawnienia do obiektu
- **RFRMQMAUT** Odśwież zabezpieczenia

Składnia tych komend oraz sposób ich użycia są opisane w pomocy do komendy CL. OAM współpracuje z *jednostką* jednostki głównej lub grupy.

Podczas wykonywania żądania MQI lub wydania komendy OAM sprawdza autoryzację jednostki powiązanej z operacją, aby sprawdzić, czy może ona wykonać następujące działania:

- Wykonaj żądaną operację.
- Uzyskaj dostęp do określonych zasobów menedżera kolejek.

Usługa autoryzacji umożliwia rozszerzanie lub zastępowanie sprawdzanych uprawnień dla menedżerów kolejek poprzez napisanie własnego komponentu usługi autoryzacji.

Definiowanie usługi w systemie operacyjnym

Sekcje usługi autoryzacji w pliku konfiguracyjnym menedżera kolejek `qm.ini` definiują usługę autoryzacji dla menedżera kolejek. Informacje na temat typów sekcji znajdują się w sekcji “Konfigurowanie usług i komponentów w systemie IBM i” na stronie 1047 .

Konfigurowanie sekcji usług autoryzacji

W systemie IBM MQ for IBM i:

Kolumnowo-wierszowa
Jest to profil użytkownika systemu IBM i .

Grupa
Jest profilem grupowym systemu IBM i .

Autoryzacje mogą być nadawane lub odbierane tylko na poziomie grupy. Żądanie nadania lub unieważnienia uprawnień użytkownika aktualizuje grupę podstawową dla tego użytkownika.

Każdy menedżer kolejek ma własny plik konfiguracyjny menedżera kolejek. Na przykład domyślna ścieżka i nazwa pliku konfiguracyjnego menedżera kolejek dla menedżera kolejek QMNAME to `/QIBM/UserData/mqm/qmgrs/QMNAME/qm.ini`.

Sekcja *`Service`* i sekcja *`ServiceComponent`* dla domyślnego komponentu autoryzacji są automatycznie dodawane do programu `qm.ini` , ale mogą zostać nadpisane przez program `WRKENVVAR`. Wszystkie pozostałe sekcje *`ServiceComponent`* należy dodać ręcznie.

Na przykład następujące sekcje w pliku konfiguracyjnym 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 105. Sekcje usługi autoryzacji w pliku qm.ini w systemie IBM i

Pierwsza sekcja komponentu usługi `MQ.UNIX.authorization.service` definiuje domyślny komponent usługi autoryzacji, czyli OAM. Jeśli ta sekcja zostanie usunięta i zrestartowany zostanie menedżer kolejek, OAM jest wyłączony i nie zostaną przeprowadzone 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 identyfikator użytkownika i hasło oraz może ustawiać pola kontekstu tożsamości.

MQZ_CHECK_AUTHORITY
Sprawdza, czy jednostka ma uprawnienia do wykonywania jednej lub większej liczby operacji na określonym obiekcie.

MQZ_COPY_ALL_AUTHORITY,
Kopiuje wszystkie bieżące autoryzacje, które istnieją dla obiektu, do którego istnieje odwołanie, do innego obiektu.

MQZ_DELETE_AUTHORITY,
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 przydzielone przydzielone zasoby.

MQZ_GET_AUTHORITY,
Pobiera uprawnienia, które jednostka musi uzyskać, aby uzyskać dostęp do określonego obiektu.

Uprawnienie MQZ_GET_EXPLICIT_AUTHORITY
Uzyskuje uprawnienie do dostępu do określonego obiektu przez nazwaną grupę (ale bez dodatkowego uprawnienia grupy **nobody**) lub uprawnienie, do którego ma dostęp grupa podstawowa nazwanego użytkownika, aby uzyskać dostęp do określonego obiektu.

MQZ_INIT_AUTHORITY,
Inicjuje komponent usługi autoryzacji.

MQZ_INQUIRE
Wysyła zapytania do obsługiwanych funkcji usługi autoryzacji.

MQZ_REFRESH_CACHE
Odśwież wszystkie autoryzacje.

MQZ_SET_AUTHORITY,
Ustawia uprawnienia, które jednostka ma do określonego obiektu.

MQZ_TERM_AUTHORITY,
Przerywa komponent usługi autoryzacji.

Te punkty wejścia obsługują użycie identyfikatora zabezpieczeń serwera Windows (NT SID).

Nazwy te są definiowane jako **typedef** w pliku nagłówkowego `cmqzc.h`, który może być używany do tworzenia prototypów funkcji komponentu.

Funkcja inicjowania (**MQZ_INIT_AUTHORITY**) musi być głównym punktem wejścia dla komponentu. Pozostałe 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 1047.

Multi Pisanie i kompilowanie wyjść funkcji API na wielu platformach

Wyjścia funkcji API umożliwiają zapisywanie kodu, który zmienia zachowanie wywołań interfejsu API produktu IBM MQ , takich jak MQPUT i MQGET, a następnie wstawiany jest ten kod bezpośrednio przed lub bezpośrednio po tych wywołaniach.

Uwaga: z/OS Nie jest obsługiwane w systemie IBM MQ for z/OS.

Dlaczego należy używać wyjść funkcji API?

Każda z aplikacji ma do wykonania konkretne zadanie, a jego kod powinien wykonać to zadanie tak efektywnie, jak to możliwe. Na wyższym poziomie może być konieczne zastosowanie standardów lub procesów biznesowych do określonego menedżera kolejek dla wszystkich aplikacji, które używają tego menedżera kolejek. Bardziej efektywne jest to, aby zrobić to powyżej poziomu poszczególnych aplikacji, a tym samym bez konieczności zmiany kodu każdej aplikacji, której dotyczy problem.

Poniżej przedstawiono kilka sugestii dotyczących obszarów, w których wyjścia funkcji API mogą być przydatne:

Zabezpieczenia
W celu zapewnienia bezpieczeństwa można zapewnić uwierzytelnianie, sprawdzanie, czy aplikacje są autoryzowane do uzyskiwania dostępu do kolejki lub menedżera kolejek. Można również policyjne aplikacje używały interfejsu API, uwierzytelniać poszczególne wywołania API, a nawet parametry, których używają.

Elastyczność
W celu zapewnienia elastyczności użytkownik może reagować na szybkie zmiany w środowisku biznesowym bez zmiany aplikacji, które polegają na danych w tym środowisku. Można na przykład użyć wyjść funkcji API, które odpowiadają na zmiany stóp procentowych, kursów wymiany walut lub ceny komponentów w środowisku produkcyjnym.

Monitorowanie korzystania z kolejki lub menedżera kolejek
W celu monitorowania użycia kolejki lub menedżera kolejek można śledzić przepływ aplikacji i komunikatów, rejestrować błędy w wywołaniach interfejsu API, skonfigurować ścieżki audytu dla celów rozliczania lub gromadzić statystyki użycia dla celów planowania.

Co się dzieje, gdy kończy się wyjście interfejsu API?

Po zapisaniu programu obsługi wyjścia i zidentyfikowaniu go w programie IBM MQ, menedżer kolejek automatycznie wywołuje kod wyjścia w zarejestrowanych punktach.

Procedury obsługi wyjścia funkcji API do uruchomienia są identyfikowane w sekcjach na następujących platformach:

- IBM i IBM i
- Linux UNIX UNIX and Linux
- Windows Windows

W tym temacie opisano sekcje w plikach konfiguracyjnych `mqs.ini` i `qm.ini`.

Definicja procedur może wystąpić w trzech miejscach:

1. ApiExitCommon, w pliku mqs.ini , identyfikuje podprogramy dla całej partycji IBM MQ, które są stosowane podczas uruchamiania menedżerów kolejek. Mogą one zostać przesłonięte przez procedury zdefiniowane dla poszczególnych menedżerów kolejek (patrz pozycja “3” na stronie 1051 na tej liście).
2. Szablon ApiExitw pliku mqs.ini identyfikuje podprogramy dla całej partycji IBM MQ, które są kopiowane do zestawu lokalnego ApiExit(patrz pozycja “3” na stronie 1051 na tej liście) podczas tworzenia nowego menedżera kolejek.
3. ApiExitLokalne, w pliku qm.ini , identyfikuje podprogramy, które mają zastosowanie do określonego 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 lokalne ApiExit, jak i ApiExit. Definicje lokalne ApiExitzastępują wspólne definicje ApiExit, jeśli obie identyfikują podprogram o tej samej nazwie. Atrybut Sequence opisany w sekcji “Konfigurowanie wyjść funkcji API” na stronie 1056 określa kolejność, w jakiej procedury zdefiniowane w sekcjach są uruchamiane.

Korzystanie z wyjść funkcji API w wielu instalacjach produktu IBM MQ

Należy upewnić się, że wyjścia funkcji API napisane dla wcześniejszej wersji produktu IBM MQ są używane do pracy ze wszystkimi wersjami, ponieważ zmiany wprowadzone do wyjść w programie IBM WebSphere MQ 7.1 mogą nie działać z wcześniejszą wersją. Więcej informacji na temat zmian wprowadzonych do wyjść zawiera sekcja “Zapisywanie wyjść i instalowalnych usług w systemach UNIX, Linux i Windows” na stronie 1033.

Przykłady udostępnione dla wyjść funkcji API amqsaem i amqsaxe odzwierciedlają zmiany wymagane podczas pisania wyjść. Aplikacja kliencka musi upewnić się, że przed uruchomieniem aplikacji powiązane są poprawne biblioteki produktu IBM MQ , które odpowiadają instalacji menedżera kolejek, z którym powiązana jest aplikacja.

Multi *Pisanie wyjść funkcji API*

Istnieje możliwość pisania wyjść dla każdego wywołania interfejsu API przy użyciu języka programowania C.

Dostępne wyjścia

Wyjścia są dostępne dla każdego wywołania interfejsu API w następujący sposób:

- MQCB, w celu ponownego zarejestrowania wywołania zwrotnego dla określonego uchwytu obiektu i aktywacji sterowania oraz zmiany w wywołaniu zwrotnym
- MQCTL do wykonywania działań sterujących na uchwytach obiektów otwartych dla połączenia
- MQCONN/MQCONNX służy do udostępniania uchwytu połączenia menedżera kolejek w celu użycia w kolejnych wywołaniach API
- MQDISC, aby odłączyć się od menedżera kolejek
- MQBEGIN, aby rozpocząć globalną jednostkę pracy (UOW)
- MQBACK, aby wycofać UOW
- MQCMIT, aby zatwierdzić jednostkę pracy
- MQOPEN, aby otworzyć zasób IBM MQ w celu późniejszego uzyskania dostępu
- MQCLOSE, aby zamknąć zasób IBM MQ , który wcześniej został otwarty dla dostępu
- MQGET-umożliwia pobranie komunikatu z kolejki, która została wcześniej otwarta na potrzeby dostępu.
- MQPUT1, aby umieścić komunikat w kolejce
- Wywołanie MQPUT w celu umieszczenia komunikatu w kolejce, która została wcześniej otwarta dla dostępu
- MQINQ, aby zapytać o atrybuty zasobu IBM MQ , który został wcześniej otwarty na potrzeby dostępu
- MQSET służy do ustawiania atrybutów kolejki, która została wcześniej otwarta na potrzeby dostępu.

- MQSTAT, w celu pobrania informacji o statusie
- MQSUB, aby zarejestrować subskrypcję aplikacji w określonym temacie
- MQSUBRQ, aby złożyć wniosek o subskrypcję

Funkcja MQ_CALLBACK_EXIT udostępnia funkcję wyjścia, która ma zostać wykonana przed i po przetworzeniu wywołania zwrotnego. Więcej informacji na ten temat zawiera sekcja Callback-MQ_CALLBACK_EXIT.

Pisanie wyjść funkcji API

W przypadku wyjść funkcji API wywołania przyjmują postać ogólną:

```
MQ_call_EXIT (parameters, context, ApiCallParameters)
```

gdzie *call* jest nazwą wywołania MQI bez przedrostka MQ ; na przykład PUT, GET. *parameters* sterują funkcją wyjścia, zapewniając przede wszystkim komunikację między wyjściem a zewnętrznymi blokami sterującą MQAXP (struktura parametru wyjścia funkcji API) i MQAXC (struktura kontekstu wyjścia funkcji API). *context* opisuje kontekst, w którym wywołano wyjście funkcji API, a program *ApiCallParameters* reprezentuje parametry wywołania MQI.

Aby ułatwić zapisanie wyjścia funkcji API, udostępniono przykładowe wyjście amqsaxe0.c; to wyjście generuje pozycje śledzenia do określonego pliku. Tego przykładu można użyć jako punktu początkowego podczas zapisywania wyjść. Więcej informacji na temat korzystania z wyjścia przykładowego zawiera sekcja “Przykładowy program obsługi wyjścia funkcji API” na stronie 1183.

Więcej informacji na temat wywołań wyjścia funkcji API, zewnętrznych bloków sterujących i powiązanych tematów zawiera sekcja Informacje dodatkowe o wyjściu interfejsu API.

Ogólne informacje na temat pisania, kompilowania i konfigurowania wyjścia zawiera sekcja “Zapisywanie wyjść i instalowalnych usług w systemach UNIX, Linux i Windows” na stronie 1033.

Korzystanie z uchwytów komunikatów w wyjściach API

Użytkownik może sterować dostępem do właściwości komunikatu, do których ma dostęp wyjście funkcji API. Właściwości są powiązane z uchwytem ExitMsg. Właściwości ustawione w wyjściu umieszczonym są ustawiane w umieszczonym komunikacie, ale właściwości pobrane w wyjściu pobierania nie są zwracane do aplikacji.

Podczas rejestrowania funkcji wyjścia MQ_INIT_EXIT za pomocą wywołania MQI MQXEP z parametrem **Function** ustawionym na wartość MQXF_INIT i **ExitReason** ustawionym na wartość MQXR_CONNECTION, należy przejść do struktury MQXEPO jako parametru **ExitOpts** . Struktura MQXEPO zawiera pole ExitProperties , które określa zestaw właściwości, które mają zostać udostępnione dla wyjścia. Jest on określany jako łańcuch znaków reprezentujący przedrostek właściwości, który odpowiada nazwie folderu MQRFH2 .

Każde wyjście funkcji API odbiera strukturę MQAXP, która zawiera pole ExitMsgHandle. To pole jest ustawiane na wartość wygenerowaną przez produkt IBM MQ i jest specyficzne dla połączenia. W związku z tym uchwyt nie zmienia się między wyjściami API tego samego lub innego typu w tym samym połączeniu.

W przypadku wartości MQ_PUT_EXIT lub MQ_PUT1_EXIT z produktem **ExitReason** z tabeli MQXR_BEFORE, czyli wyjścia interfejsu API wykonywanego przed umieszczniem komunikatu, wszystkie właściwości (inne niż właściwości deskryptora komunikatu) powiązane z uchwytem ExitMsgpo zakończeniu wyjścia są ustawiane w umieszczonym komunikacie. Aby temu zapobiec, należy ustawić parametr Uchwyt ExitMsgna wartość MQHM_NONE. Można także podać inny uchwyt komunikatu.

W przypadku operacji MQ_GET_EXIT i MQ_CALLBACK_EXIT Uchwyt ExitMsgjest czyszczony z właściwości i zapełniany właściwościami określonymi w polu ExitProperties , gdy zarejestrowano wartość MQ_INIT_EXIT, inne niż właściwości deskryptora komunikatu. Właściwości te nie są dostępne dla aplikacji pobierających. Jeśli aplikacja pobierający określiła uchwyt komunikatu w polu MQGMO (Get message options), to wszystkie właściwości powiązane z tym uchwytem, w tym właściwości

deskryptora komunikatu, są dostępne dla wyjścia funkcji API. Aby zapobiec zapełnieniu uchwytu ExitMsgwłaściwościami, należy ustawić go na wartość MQHM_NONE.

Uwaga: Dla właściwości komunikatu wyjścia, które mają zostać przetworzone w:

- Po funkcji MQ_GET_EXIT konieczne jest zdefiniowanie przed wyjściem funkcji MQ_GET_EXIT.
- Przed funkcją MQ_CALLBACK_EXIT należy zdefiniować wartość przed funkcją MQ_CB_EXIT dla wyjścia.

Dostępny jest przykładowy program amqsaem0.cilustrujący użycie uchwytów komunikatów w wyjściach API.

Odsyłacze pokrewne

Procedury zewnętrzne, wyjścia funkcji API i odwołania do usług instalowalnych

Multi *Kompilowanie wyjść funkcji API*

Po zapisaniu wyjścia należy skompilować go i połączyć w następujący sposób.

W poniższych przykładach przedstawiono komendy używane dla przykładowego programu opisanego w sekcji “Przykładowy program obsługi wyjścia funkcji API” na stronie 1183. W przypadku platform innych niż Windows można znaleźć przykładowy kod wyjścia funkcji API w programie *MQ_INSTALLATION_PATH*/samp oraz skompilowaną i dowiązaną bibliotekę współużytkowaną w programie *MQ_INSTALLATION_PATH*/samp/bin.

Windows W przypadku systemów Windows przykładowy kod wyjścia funkcji API można znaleźć w programie *MQ_INSTALLATION_PATH* \Tools\c\Samples. *MQ_INSTALLATION_PATH* reprezentuje katalog, w którym zainstalowano produkt IBM MQ .

Uwaga: Wskazówki dotyczące programowania aplikacji 64-bitowych są wymienione w sekcji Standardy kodowania na platformach 64-bitowych.

W przypadku klientów rozsyłania grupowego wyjścia funkcji API i wyjścia konwersji danych muszą być możliwe do uruchomienia po stronie klienta, ponieważ niektóre komunikaty mogą nie przechodzić przez menedżer kolejek. Następujące biblioteki są częścią pakietów klienta, a także pakietów serwera:

Tabela 145. Biblioteki, które znajdują się w pakietach klienta i serwera

System operacyjny	Biblioteki
AIX AIX	32 bity & 64 bit: libmqm.a & libmqm_r.a
IBM i IBM i	LIBMQM & LIBMQM_R
Linux Linux	32 bity & 64 bit: libmqm.so & libmqm_r.so
Solaris Solaris	32 bity & 64 bit: libmqm.so
Windows Windows	32 bity & 64 bit: mqm.dll & mqm.pdb

Linux UNIX *Kompilowanie wyjść funkcji API w systemach UNIX i Linux*

Przykłady sposobu kompilowania wyjść funkcji API w systemach UNIX and Linux .

Na wszystkich platformach punktem wejścia do modułu jest MQStart.

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

wł.AIX

AIX

Skompiluj kod źródłowy wyjścia funkcji API, wydając jedną z następujących komend:

Aplikacje 32-bitowe

Niewątkowa

```
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

Niewątkowa

```
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:

31-bitowe aplikacje

Niewątkowa

```
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

Niewątkowa

```
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

Niewątkowa

```
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
```

wł.Solaris

Solaris

Skompiluj kod źródłowy wyjścia funkcji API, wydając jedną z następujących komend:

Aplikacje 32-bitowe

platforma SPARC

```
cc -xarch=v8plus -KPIC -mt -G -o /var/mqm/exits/amqsaxe \
   amqsaxe0.c -I MQ_INSTALLATION_PATH/inc \
   -R/usr/lib/32 -lsocket -lnsl -ldl
```

platformax86-64

```
cc -xarch=386 -KPIC -mt -G -o /var/mqm/exits/amqsaxe \
   amqsaxe0.c -I MQ_INSTALLATION_PATH/inc \
   -R/usr/lib/32 -lsocket -lnsl -ldl
```

Aplikacje 64-bitowe

platforma SPARC

```
cc -xarch=v9 -KPIC -mt -G -o /var/mqm/exits64/amqsaxe \
   amqsaxe0.c -I MQ_INSTALLATION_PATH/inc \
   -R/usr/lib/64 -lsocket -lnsl -ldl
```

platformax86-64

```
cc -xarch=amd64 -KPIC -mt -G -o /var/mqm/exits64/amqsaxe \
   amqsaxe0.c -I MQ_INSTALLATION_PATH/inc \
   -R/usr/lib/64 -lsocket -lnsl -ldl
```

Windows *Kompilowanie wyjść funkcji API w systemach Windows*

Skompiluj i dowiąz 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 taki sam, należy pominąć parametr -manifest *manifest.file* w komendzie **mt** .

Dostosuj komendy podane w przykładach w produkcie Rysunek 106 na stronie 1056 lub Rysunek 107 na stronie 1056 , aby skompilować i dowiązac `amqsaxe0.c` w systemie Windows. Komendy działają z produktem Microsoft Visual Studio 2008, 2010 lub 2012. W przykładach założono, że katalog `C:\Program Files\IBM\MQ\tools\c\samples` jest bieżącym katalogiem.

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 106. Kompiluj i dowiąże 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 107. Skompiluj i dowiąże amqsaxe0.c w 64-bitowym systemie Windows

Pojęcia pokrewne

“Przykładowy program obsługi wyjścia funkcji API” na stronie 1183
Przykładowe wyjście interfejsu API generuje dane śledzenia MQI w pliku określonym przez użytkownika z przedrostkiem zdefiniowanym w zmiennej środowiskowej MQAPI_TRACE_LOGFILE.

IBM i *Komplementowanie wyjść funkcji API w systemie IBM i*

Kompilowanie wyjść funkcji API w systemie IBM i.

Wyjście jest tworzone w następujący sposób (dla przykładu języka C):

1. Utwórz moduł przy użyciu komendy CRTCMOD. Skompiluj go, aby użyć teraprzestrzeni, włączając w to parametr TERASPACE(*YES *TSIFC).
2. Utwórz program usługowy z modułu za pomocą komendy CRTSRVPGM. Należy powiązać go z programem usługowym QMQM/LIBMQMZF_R w celu obsługi wielowątkowych wyjść funkcji API.

Konfigurowanie wyjść funkcji API

Aby włączyć wyjścia funkcji API, należy skonfigurować produkt IBM MQ , zmieniając informacje o konfiguracji.

Aby zmienić informacje konfiguracyjne, należy zmienić sekcje definiujące procedury obsługi wyjścia i sekwencję, w której są one uruchamiane. Informacje te mogą zostać zmienione w następujący sposób:

- Windows Linux Using the IBM MQ Explorer on Windows and Linux (x86 and x86-64 platforms).
- Windows Za pomocą komendy **amqmdain** w systemie Windows.

- Multi Korzystanie z plików `mqs.ini` i `qm.ini` bezpośrednio w systemach Windows, IBM i IBM i, i UNIX and Linux .

  Plik `mqs.ini` zawiera informacje istotne dla wszystkich menedżerów kolejek w określonym węźle. Można go znaleźć w następujących lokalizacjach:

  – IBM i W katalogu `/QIBM/UserData/mqm` na serwerze IBM i.
  – Linux UNIX W katalogu `/var/mqm` na serwerze UNIX and Linux.
  – Windows W ścieżce WorkPath określonej w kluczu `HKLM\SOFTWARE\IBM\WebSphere MQ` w systemach Windows .

  Plik `qm.ini` zawiera informacje istotne dla konkretnego menedżera kolejek. Dla każdego menedżera kolejek znajduje się 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 UNIX W systemach UNIX and Linux :

  ```
  /var/mqm/qmgrs/QMNAME/qm.ini
  ```

  Windows W systemach Windows :

  ```
  C:\ProgramData\IBM\MQ\qmgrs\QMNAME\qm.ini
  ```

  Przed edytowaniem pliku konfiguracyjnego należy utworzyć kopię zapasową w taki sposób, aby możliwe było przywrócenie kopii, jeśli zajdziesz taka potrzeba.

  Pliki konfiguracyjne można edytować w jeden z następujących sposobów:

  – Automatycznie, przy użyciu komend, które zmieniają konfigurację menedżerów kolejek w węźle.
  – Ręcznie, przy użyciu standardowego edytora tekstu.

  Jeśli w atrybucie pliku konfiguracyjnego zostanie ustawiona niepoprawna wartość, wartość ta zostanie zignorowana i zostanie wyświetlony komunikat operatora wskazujący problem. Efekt jest taki sam, jak brak w całości atrybutu.

Sekcje do skonfigurowania

Sekcje, które muszą zostać zmienione, są następujące:

ApiExitCommon

Zdefiniowane w składach `mqs.ini` i IBM MQ Explorer na stronie właściwości produktu IBM MQ , w sekcji Exits (Wyjmowana).

Po uruchomieniu menedżera kolejek atrybuty znajdujące się w tej sekcji są odczytyywane, a następnie nadpisywane przez wyjścia interfejsu API zdefiniowane w programie `qm.ini`.

ApiExitTemplate

Zdefiniowane w składach `mqs.ini` i IBM MQ Explorer na stronie właściwości produktu IBM MQ , w sekcji Exits (Wyjmowana).

Gdy tworzony jest dowolny menedżer kolejek, atrybuty w tej sekcji są kopiowane do nowo utworzonego pliku qm.ini w sekcji ApiExitLocal.

ApiExitLocal

Zdefiniowane w składach `qm.ini` i IBM MQ Explorer na stronie właściwości menedżera kolejek, w sekcji Exits (Wyjmowana).

Po uruchomieniu menedżera kolejek wyjścia funkcji API zdefiniowane w tym miejscu przesłaniają wartości domyślne zdefiniowane w produkcie `mqs.ini`.

Atrybuty sekcji

- Podaj nazwę wyjścia funkcji API, używając następującego atrybutu:

  Name=nazwa_ApiExit_name
  Opisowa nazwa wyjścia interfejsu API przekazana do niego w polu ExitInfow polu Nazwa struktury MQAXP.

  Ta nazwa musi być unikalna, nie dłuższa niż 48 znaków i zawierać tylko poprawne znaki dla nazw obiektów IBM MQ (na przykład nazwy kolejek).

- Zidentyfikuj moduł i punkt wejścia kodu wyjścia API, który ma być uruchamiany przy użyciu następujących atrybutów:

  Function=nazwa_funkcji
  Nazwa punktu wejścia funkcji do modułu zawierającego kod wyjścia API. Ten punkt wejścia jest funkcją MQ_INIT_EXIT.

  Wielkość tego pola jest ograniczona do wartości MQ_EXIT_NAME_LENGTH.

  Module=nazwa_modułu
  Moduł zawierający kod wyjścia API.

  Jeśli w polu znajduje się pełna nazwa ścieżki do modułu, jest ona używana w takiej postaci.

  Jeśli to pole zawiera tylko nazwę modułu, to moduł znajduje się przy użyciu atrybutu `ExitsDefaultPath` w pliku `ExitPath` w pliku qm.ini.

  Na platformach obsługujących oddzielne biblioteki wielowątkowe należy udostępnić wersję modułu wyjścia API, która nie jest wielowątkowa i jest wielowątkowa. Wersja wielowątkowa musi mieć przyrostek `_r` . Wersja wielowątkowa kodu pośredniczącego aplikacji IBM MQ niejawnie dopisuje `_r` do podanej nazwy modułu przed jego załadowaniu.

  Długość tego pola jest ograniczona do maksymalnej długości ścieżki, którą obsługuje platforma.

- Opcjonalnie przekaz dane za pomocą wyjścia za pomocą następującego atrybutu:

  Data=nazwa_danych
  Dane, które mają być przekazywane do wyjścia funkcji API w polu ExitData struktury MQAXP.

  Jeśli zostanie podany ten atrybut, początkowe i końcowe odstępy zostaną usunięte, pozostały łańcuch zostanie obcięty do 32 znaków, a wynik zostanie przekazany do wyjścia. Jeśli ten atrybut zostanie pominięty, do wyjścia zostanie przekazana domyślna wartość 32 odstępów.

  Maksymalna długość tego pola to 32 znaki.

- Zidentyfikuj sekwencję tego wyjścia w odniesieniu do innych wyjść, korzystając z następującego atrybutu:

  Sequence=numer_sekwencji_
  Sekwencja, w której to wyjście funkcji API jest wywoływane w stosunku do 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 jednoznaczna. Sekwencja 1, 2, 3 ma ten sam wynik co sekwencja 7, 42, 1096. Jeśli dwa wyjścia mają ten sam numer kolejny, menedżer kolejek decyduje o tym, który z nich powinien najpierw zadzwonić. Można określić, który został wywołany po zdarzeniu, umieszczając czas lub znacznik w obszarze ExitChain(Łańcuch

ExitChain) wskazany przez ExitChainAreaPtr w produkcie MQAXP lub przez zapisanie własnego pliku dziennika.

Ten atrybut jest niepodpisaną wartością liczbową.

Sekcje przykładowe

Przykładowy plik mqs.ini zawiera następujące sekcje:

ApiExitTemplate
Ta sekcja definiuje wyjście z nazwą opisową `OurPayrollQueueAuditor`, nazwą modułu `auditor`i numerem kolejnym 2. Do wyjścia jest przekazywana wartość danych o wartości 123.

ApiExitCommon
Ta sekcja definiuje wyjście z nazwą opisową `MQPoliceman`, nazwą modułu `tmqp`i numerem kolejnym 1. Przekazywane dane są instrukcjami (`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 ApiExitwyjścia o 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
```

Programy obsługi wyjścia kanału dla kanałów przesyłania komunikatów

Ta kolekcja tematów zawiera informacje na temat programów obsługi wyjścia kanału produktu IBM MQ dla kanałów przesyłania komunikatów.

Agenty kanałów komunikatów (MCAs) mogą również wywoływać wyjścia konwersji danych. Więcej informacji na temat pisania wyjść konwersji danych zawiera sekcja “Pisanie wyjść konwersji danych” na stronie 1082.

Niektóre z tych informacji mają zastosowanie również do wyjść z kanałów 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 obsługi wyjścia kanału są wywoływane w zdefiniowanych miejscach w przetwarzaniu wykonywanego przez programy MCA.

Niektóre z tych programów obsługi wyjścia użytkownika działają w pary komplementarne. Na przykład, jeśli program obsługi wyjścia użytkownika jest wywoływany przez wysyłający agent MCA w celu zaszyfrowania komunikatów do transmisji, proces komplementarny musi działać w odbiorczym końcu, aby odwrócić ten proces.

W programie Tabela 146 na stronie 1060 wyświetlane są typy wyjść kanału, które są dostępne dla każdego typu kanału.

Tabela 146. Wyjścia kanału dostępne dla każdego typu kanału						
Typ kanału	**Wyjście komunikatu**	**Wyjście z ponowienia komunikatu**	**Wyjście odbierania**	**Wyjście zabezpieczeń**	**Wyjście wysyłania**	**Wyjście automatycznej definicji**
Kanał nadawcy	Tak		Tak	Tak	Tak	
Kanał serwera	Tak		Tak	Tak	Tak	
Kanał wysyłający klastry	Tak		Tak	Tak	Tak	Tak
Kanał odbiorcy	Tak	Tak	Tak	Tak	Tak	Tak
Kanał requestera	Tak	Tak	Tak	Tak	Tak	
Kanał odbierający klastry	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/OSwyjście definicji auto-definition dotyczy tylko kanałów wysyłających klastry i kanały odbierające klastry.

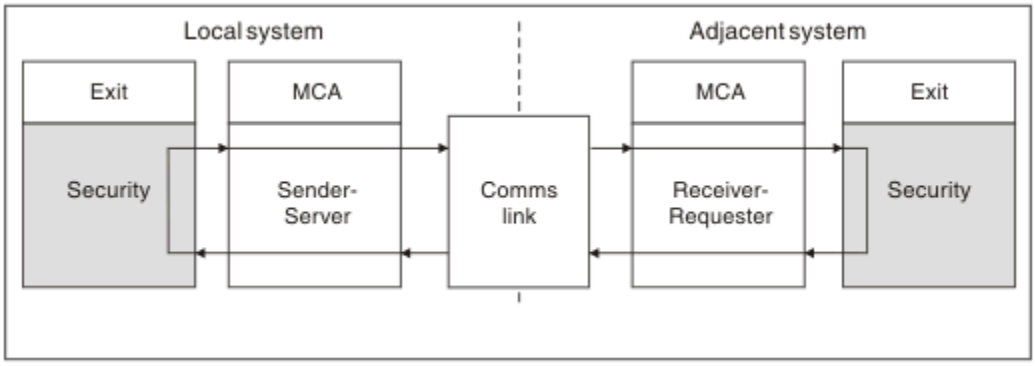
Jeśli użytkownik zamierza uruchomić wyjścia kanału na kliencie, nie można użyć zmiennej środowiskowej MQSERVER. Zamiast tego należy utworzyć i odwołać się do tabeli definicji kanału klienta (CCDT) zgodnie z opisem w sekcji Tabela definicji kanału klienta.

Przegląd przetwarzania

Przegląd informacji o tym, w jaki sposób MCAs korzysta z programów obsługi wyjścia.

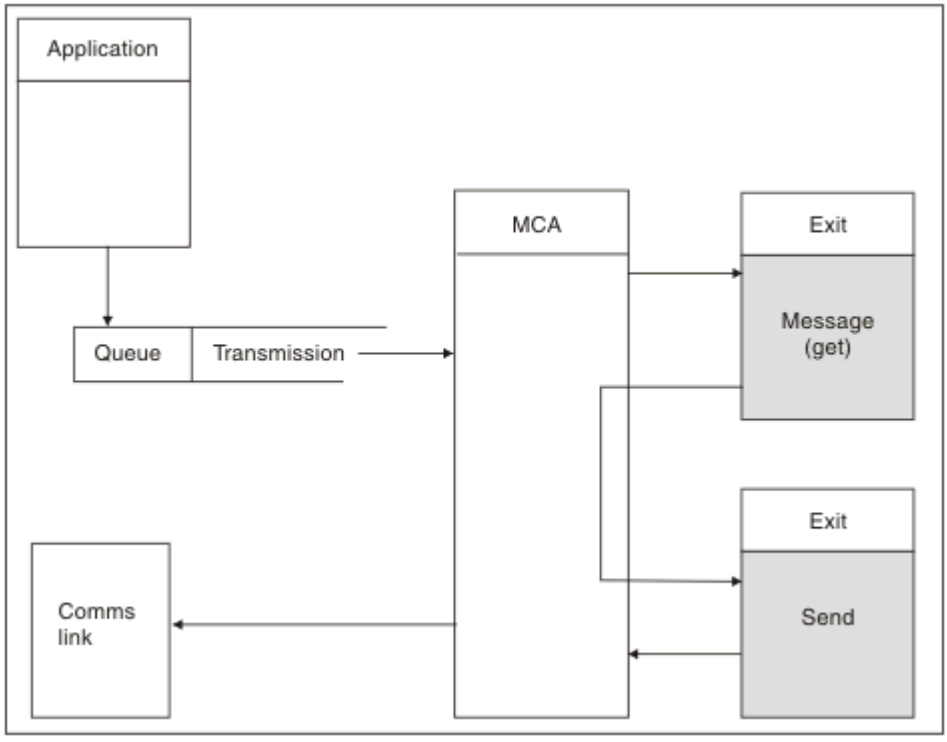
Podczas uruchamiania, MCAs wymieniają okno dialogowe uruchamiania w celu zsynchronizowania przetwarzania. Następnie przełączają się na wymianę danych, która obejmuje wyjścia zabezpieczeń. Te wyjścia muszą zakończyć się pomyślnie, aby faza uruchamiania zakończyła się pomyślnie i aby umożliwić przesyłanie komunikatów.

Faza sprawdzania zabezpieczeń jest pętlą, jak pokazano na Rysunek 108 na stronie 1061.

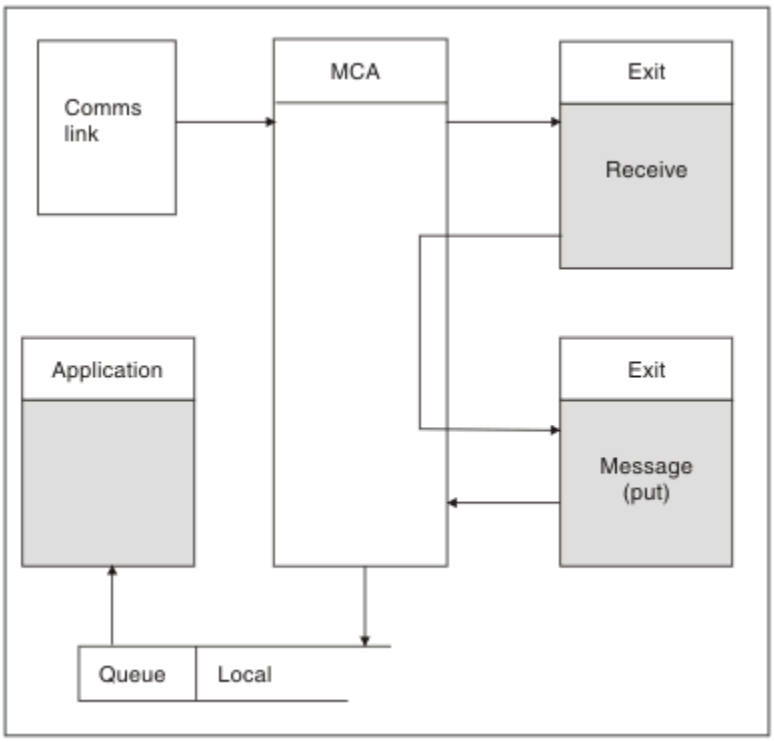


Rysunek 108. Pętla wyjścia zabezpieczeń

Podczas fazy przesyłania komunikatów wysyłający agent MCA pobiera komunikaty z kolejki transmisji, wywołuje wyjście komunikatów, wywołuje wyjście wysyłania, a następnie wysyła komunikat do odbierającego agenta MCA, jak to pokazano na rysunku Rysunek 109 na stronie 1061.



Rysunek 109. Przykład wyjścia wysyłania na końcu kanału komunikatów nadawcy



Rysunek 110. Przykład wyjścia odbierania na końcu kanału komunikatów kanału komunikatów

Odbierający agent MCA odbierze komunikat z łącza komunikacyjnego, wywołuje wyjście odbierania, wywołuje wyjście komunikatu, a następnie umieszcza komunikat w kolejce lokalnej, jak to pokazano na Rysunek 110 na stronie 1062. (Wyjście odbierania może być wywoływane więcej niż jeden raz przed wywołaniem wyjścia komunikatu).

Pisanie programów obsługi wyjścia kanału

W celu ułatwienia pisania programów obsługi wyjścia kanału można użyć następujących informacji.

Programy zewnętrzne i programy obsługi wyjścia kanału mogą korzystać ze wszystkich wywołań MQI, z wyjątkiem sytuacji, w których są one oznaczone w kolejnych sekcjach. W przypadku produktu MQ V7 i nowszego struktura MQCXP w wersji 7 i nowszej zawiera uchwyt połączenia hConn, który może zostać użyty zamiast wywołania MQCONN. W przypadku wcześniejszych wersji, aby uzyskać uchwyt połączenia, należy wydać komendę MQCONN, nawet jeśli zostanie zwrócone ostrzeżenie MQRC_ALREADY_CONNECTED, ponieważ sam kanał jest połączony z menedżerem kolejek.

Należy pamiętać, że wyjście kanału musi być wątkowo bezpieczne.

W przypadku wyjść z kanałów połączenia klienckiego menedżer kolejek, z którym program obsługi wyjścia próbuje nawiązać połączenie, zależy od sposobu, w jaki połączenie zostało nawiązane. Jeśli wyjście zostało połączone z programem MQM.LIB (lub QMQM/LIBMQM w systemie IBM i) i nie określono nazwy menedżera kolejek w wywołaniu MQCONN, program zewnętrzny próbuje połączyć się z domyślnym menedżerem kolejek w systemie. Jeśli wyjście zostało połączone z programem MQM.LIB (lub QMQM/LIBMQM w systemie IBM i) i określa nazwę menedżera kolejek, który został przekazany do wyjścia za pomocą pola QMgrName produktu MQCD, wyjście próbuje nawiązać połączenie z tym menedżerem kolejek. Jeśli wyjście było połączone z MQIC.LIB lub dowolna inna biblioteka, wywołanie MQCONN nie powiedzie się, niezależnie od tego, czy zostanie określona nazwa menedżera kolejek.

Należy unikać zmiany stanu transakcji powiązanej z przekazanej wartości hConn w wyjściu kanału. Nie wolno używać komend MQCMIT, MQBACK ani MQDISC z kanałem hConn, a komendy MQBEGIN nie można używać z określeniem kanału hConn.

Jeśli parametr MQCONNX jest używany z parametrem MQCNO_HANDLE_SHARE_BLOCK lub MQCNO_HANDLE_SHARE_NO_BLOCK w celu utworzenia nowego połączenia z produktem IBM MQ ,

należy upewnić się, że połączenie jest poprawnie zarządzane i poprawnie rozłącza się z menedżerem kolejek. Na przykład: wyjście kanału, które tworzy nowe połączenie z menedżerem kolejek w każdym wywołaniu bez rozłączania, powoduje, że uchwyty połączeń są tworzone i zwiększane w liczbie wątków agenta.

Wyjście jest uruchamiane w tym samym wątku co sam agent MCA i korzysta z tego samego uchwytu połączenia. Dlatego jest on uruchamiany w obrębie tej samej jednostki pracy, co agent MCA, a wszystkie wywołania wykonane w punkcie synchronizacji są zatwierdzane lub wycofane przez kanał na końcu zadania wsadowego.

Oznacza to, że wyjście komunikatu kanału może wysyłać komunikaty powiadomienia, które są zatwierdzane tylko do tej kolejki po zatwierdzeniu zadania wsadowego zawierającego oryginalną wiadomość. Możliwe jest więc wystawianie wywołań MQI punktów synchronizacji z wyjścia komunikatów kanału.

Wyjście kanału może zmienić pola na zmaterializowanych tabelach MQCD. Zmiany te nie są jednak wykonywane, z wyjątkiem wymienionych okoliczności. Jeśli program obsługi wyjścia kanału zmienia pole w strukturze danych MQCD, nowa wartość jest ignorowana przez proces kanału IBM MQ . Nowa wartość pozostaje jednak na zmaterializowanych tabelach MQCD i jest przekazywana do pozostałych wyjść w łańcuchu wyjścia i do dowolnej 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 funkcja biblioteki C bez ponownego wejścia nie może być używana w programie obsługi wyjścia kanału.

Linux UNIX Jeśli jednocześnie używane jest wiele bibliotek wyjścia kanału, mogą wystąpić problemy na niektórych platformach UNIX and Linux , jeśli kod dla dwóch różnych wyjść zawiera identycznie nazwane funkcje. Po załadowaniu wyjścia kanału program ładujący dynamiczny rozstrzywa nazwy funkcji w bibliotece wyjścia na adresy, na których załadowana jest biblioteka. Jeśli dwie biblioteki wyjścia definiują oddzielne funkcje, które zdarzają się, że mają identyczne nazwy, ten proces rozstrzygania może niepoprawnie przetłumaczać nazwy funkcji jednej biblioteki w celu użycia funkcji innego. Jeśli ten problem wystąpi, należy określić dla konsolidatora, że musi on wyeksportować tylko wymagane funkcje wyjścia i wywołania MQStart, ponieważ te funkcje nie mają wpływu na działanie. Inne funkcje muszą być podane w lokalnej widoczności, tak aby nie były używane przez funkcje poza własną biblioteką wyjścia. Aby uzyskać więcej informacji, zapoznaj się z dokumentacją urządzenia konsolidatora.

Wszystkie wyjścia są wywoływane za pomocą struktury parametru wyjścia kanału (MQCXP), struktury definicji kanału (MQCD), przygotowanego buforu danych, parametru długości danych i parametru długości buforu. Długość buforu nie może być przekroczona:

- W przypadku wyjść komunikatów należy zezwolić na wysłanie największego komunikatu, który ma być wysyłany przez kanał, powiększony o długość struktury MQXQH.
- W przypadku wyjścia wysyłania i odbierania największy bufor musi być dostępny w następujący sposób:

  LU 6.2
  32 kB

  TCP:
  IBM i IBM i 16 kB

  IBM i Inne 32 kB

  Uwaga: Maksymalna długość użyteczna może być mniejsza o 2 bajty niż długość tej długości. Sprawdź wartość zwróconej w polu MaxSegment, aby uzyskać szczegółowe informacje. Więcej informacji na temat długości MaxSegmentmożna znaleźć w sekcji MaxSegmentLength.

  NetBIOS:
  64 kB

  SPX:
  64 kB

Uwaga: Wyjścia odbierania w kanałach nadawczych i wyjściach nadawczych w kanałach odbiorczych korzystają z buforów o wielkości 2 kB dla TCP.

- W przypadku wyjść zabezpieczeń rozproszona funkcja kolejkowania przydziela bufor o wielkości 4000 bajtów.

Dopuszczalne jest zwrócenie innego bufora, wraz z odpowiednimi parametrami. Więcej informacji na ten temat zawiera sekcja “Programy obsługi wyjścia kanału dla kanałów przesyłania komunikatów” na stronie 1059 .

z/OS *Pisanie programów obsługi wyjścia kanału w systemie z/OS*

Poniższe informacje umożliwiają pisanie i kompilowanie programów obsługi wyjścia kanału dla produktu z/OS.

Wyjścia są uruchamiane tak, jakby przez odsyłacz z/OS , w:

- Nieautoryzowany stan programu problemu
- Podstawowy tryb sterowania przestrzenią adresową
- Tryb niepamięciowy
- Tryb rejestru bez dostępu
- 31-bitowy tryb adresowania

Moduły edytowane łączem muszą być umieszczone w zestawie danych określonym przez instrukcję CSQXLIB DD procedury przestrzeni adresowej inicjatora kanału; nazwy modułów ładowania są określane jako nazwy wyjścia w definicji kanału.

Podczas zapisywania wyjść kanału dla produktu z/OSmają zastosowanie następujące reguły:

- Wyjścia muszą być napisane w asemblerze lub w języku C; jeśli używany jest język C, musi być on zgodny ze środowiskiem programistycznym systemów w języku C dla wyjść systemowych, opisane w publikacji z/OS C/C++ Programming Guide.
- Wyjścia są ładowane z nieautoryzowanych bibliotek zdefiniowanych przez instrukcję CSQXLIB DD. Jeśli CSQXLIB ma DISP=SHR, wyjścia mogą być aktualizowane, gdy inicjator kanału jest uruchomiony. Nowa wersja jest używana, gdy kanał jest restartowany.
- Wyjścia muszą być ponownie uruchamiane i zdolne do działania w dowolnym miejscu wirtualnej pamięci masowej.
- Wyjścia muszą resetować środowisko, po powrocie, do tego w momencie wejścia.
- Wyjścia muszą zwolnić dowolną ilość pamięci masowej lub zapewnić, że jest ona zwalniana przez kolejne wywołanie wyjścia.

  W przypadku pamięci masowej, która ma zostać utrzymana między wywołaniami, należy użyć usługi z/OS STORAGE lub 4kmalc w celu programowania w systemie C.

  Więcej informacji na temat tej funkcji można znaleźć w sekcji 4kmalc() -- Allocate Page-Aligned Storage.
- Można użyć wszystkich wywołań MQI produktu IBM MQ z wyjątkiem komend 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 edytowane przy użyciu odsyłacza do kodu pośredniczącego CSQXSTUB.

  Wyjątkiem od tej reguły jest to, że wyjścia kanału zabezpieczeń mogą wydawać wywołania MQI dotyczące zatwierdzania i wycofywania. Aby wywołać takie wywołania, należy zakodować czasowniki CSQXCMT i CSQXBAK w miejsce MQCMIT lub CSQBCMT i MQBACK lub CSQBBAK.
- Wszystkie wyjścia, które używają kodu pośredniczącego CSQXSTUB z produktu IBM WebSphere MQ 7.0 lub nowszego, muszą być edytowane za pomocą odsyłaczy w bibliotece ładowania CSQXLIB o formacie PDS-E.
- Wyjścia nie mogą używać żadnych usług systemowych, które powodują oczekiwanie, ponieważ korzystanie z usług systemowych poważnie wpłynie na obsługę niektórych lub wszystkich innych kanałów. Wiele kanałów jest zwykle uruchamianych w pojedynczej bazie TCB. Jeśli użytkownik zrobi coś w wyjściu, który powoduje oczekiwanie, a nie zostanie użyty program MQXWAIT, to wszystkie te

kanały będą czekać. Powoduje, że kanały oczekiwania nie mają problemów z funkcjonalnością, ale mogą mieć negatywny wpływ na wydajność. Większość SVC wymaga oczekiwania, więc należy unikać ich, z wyjątkiem następujących SVC:

- GETMAIN/FREEMAIN/STORAGE
- LOAD/DELETE

W związku z tym należy unikać SVC, komputerów PC i urządzeń we/wy. Zamiast tego należy użyć wywołania MQXWAIT.

- Wyjścia nie wydają ESTAEs ani SPIES, oprócz wszystkich podzadań, które dołączają, ponieważ ich obsługa błędów może kolidować z obsługą błędów wykonywaną przez produkt 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 operacje we/wy i inne zdarzenia. Jeśli ta usługa jest używana, wyjścia nie mogą używać stosu połączeń.

  W przypadku urządzeń we/wy i innych urządzeń, które nie udostępniają obiektów nieblokujących lub EBC oczekuje na oczekiwanie, osobne podzadanie musi być typu ATTACHed, a jego zakończenie oczekiwane przez MQXWAIT; ze względu na przetwarzanie, które ta technika ma wartość incurs, funkcja ta musi być używana tylko przez wyjście zabezpieczeń.

- Wywołanie MQI MQDISC nie powoduje niejawnego zatwierdzania w programie obsługi wyjścia. Zatwierdzenie procesu kanału jest wykonywane tylko wtedy, gdy protokół kanału jest dyktowany.

Następujące przykłady wyjścia są dostarczane wraz z produktem IBM MQ for z/OS:

CSQ4BAX0
Ten przykład jest napisany w asemblerze i obrazuje użycie komendy MQXWAIT.

CSQ4BCX1 i CSQ4BCX2
Przykłady te są napisane w języku C i ilustrują sposób uzyskiwania dostępu do parametrów.

CSQ4BCX3 i CSQ4BAX3
Te przykłady są zapisywane odpowiednio w języku C i asemblerze.

Przykład CSQ4BCX3 (wstępnie skompilowany do biblioteki LOADLIB SCSQAUTH) powinien działać bez konieczności wprowadzania zmian w samym wyjściu. Istnieje możliwość utworzenia parametru LOADLIB (na przykład MY.TEST.LOADLIB) i skopiuj do niego element SCSQAUTH (CSQ4BCX3).

Aby skonfigurować wyjście zabezpieczeń dla połączenia klienckiego, wykonaj następującą procedurę:

1. Określ poprawny segment OMVS dla ID użytkownika używanego przez inicjatora kanału.

   Dzięki temu inicjator kanału IBM MQ for z/OS może używać protokołu TCP/IP z interfejsem gniazda USS (UNIX System Services), aby ułatwić przetwarzanie wyjścia. Należy pamiętać, że nie jest konieczne definiowanie segmentu OMVS dla ID użytkownika dowolnego klienta łączącego.
2. Upewnij się, że sam kod wyjścia jest uruchamiany tylko w środowisku kontrolowanym przez program.

   Oznacza to, że wszystkie załadowane do przestrzeni adresowej CHINIT muszą być załadowane z biblioteki sterowanej przez program (co oznacza wszystkie biblioteki w bibliotece STEPLIB), a wszystkie biblioteki o nazwach w bibliotece CSQXLIB i

   ```
   ++hlq++.SCSQANLx
   ++hlq++.SCSQMVR1
   ++hlq++.SCSQAUTH
   ```

   Aby ustawić bibliotekę ładowania jako sterowaną przez program, należy użyć komendy podobnej do tej w następującym przykładzie:

   ```
   RALTER PROGRAM * ADDMEM('MY.TEST.LOADLIB'//NOPADCHK)
   ```

Następnie można aktywować lub odświeżać środowisko kontrolowane przez program, wydając komendę:

```
SETROPTS WHEN(PROGRAM) REFRESH
```

3. Dodaj wyjście LOADLIB do DD CSQXLIB (w procedurze uruchomionej CHINIT), wydając następującą komendę:

```
ALTER CHANNEL(xxxx) CHLTYPE(SVRCONN)SCYEXIT(CSQ4BCX3)
```

   Powoduje to aktywowanie wyjścia dla kanału nazwanego.

4. Zewnętrzny menedżer zabezpieczeń (ESM) zawiera listę wszystkich innych bibliotek, które mają być kontrolowane przez program, ale należy pamiętać, że żadna z bibliotek ESM lub C nie musi być sterowana programem.

Więcej informacji na temat konfigurowania wyjścia zabezpieczeń przy użyciu przykładowego CSQ4BCX3zawiera sekcja IBM MQ for z/OS kanał połączenia z serwerem .

CSQ4BCX4

Ten przykład jest zapisywany w języku C i demonstruje za pomocą pól **RemoteProduct** i **RemoteVersion** w produkcie MQCXP.

Pojęcia pokrewne

IBM MQ for z/OS Kanał połączenia z serwerem

“Pisanie programów obsługi wyjścia kanału w systemie IBM i” na stronie 1066

Poniższe informacje umożliwiają pisanie i kompilowanie programów obsługi wyjścia kanału dla produktu IBM i.

“Pisanie programów obsługi wyjścia kanału w systemie UNIX, Linux, and Windows” na stronie 1067

Poniższe informacje ułatwiają pisanie programów obsługi wyjścia kanału dla systemów UNIX, Linux, and Windows .

IBM i *Pisanie programów obsługi wyjścia kanału w systemie IBM i*

Poniższe informacje umożliwiają pisanie i kompilowanie programów obsługi wyjścia kanału dla produktu IBM i.

Wyjście jest obiektem programu napisanego w języku ILE C, ILE RPG lub ILE COBOL. Nazwy programów obsługi wyjścia i ich biblioteki zostały nazwane 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 ILE COBOL należy określić SERIALIZE dla opcji THREAD instrukcji PROCESS. Programy te muszą również być powiązane z bibliotekami w wersji IBM MQ : QMQM/LIBMQM_R w przypadku ILE C i ILE RPG, oraz AMQ0STUB_R w przypadku ILE COBOL. Dodatkowe informacje na temat bezpiecznego wątku aplikacji w języku RPG lub COBOL można znaleźć w odpowiednim podręczniku programisty dla języka.
- Program IBM MQ for IBM i wymaga, aby programy obsługi wyjścia były włączone dla obsługi teraprzestrzeni. (Teraspace 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 nowsze są tak włączone. W przypadku 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 wskazany obiekt istnieje poza zakresem czasu programu obsługi wyjścia kanału. Wskaźnik nie może być adresem zmiennej na stosie programu ani zmienną w stercie programu. Zamiast tego wskaźnik musi być uzyskany z systemu. Przykładem może być przestrzeń użytkownika utworzona w wyjściu użytkownika. Aby upewnić się, że wszystkie obszary danych przydzielone przez program obsługi wyjścia kanału są nadal dostępne

dla agenta MCA po zakończeniu działania programu, wyjście kanału musi być uruchomione w grupie aktywacji programu wywołującego lub nazwanej grupy aktywacji. W tym celu należy ustawić parametr ACTGRP w parametrze 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 UNIX, Linux, and Windows” na stronie 1067
Poniższe informacje ułatwiają pisanie programów obsługi wyjścia kanału dla systemów UNIX, Linux, and Windows .

“Pisanie programów obsługi wyjścia kanału w systemie z/OS” na stronie 1064
Poniższe informacje umożliwiają pisanie i kompilowanie programów obsługi wyjścia kanału dla produktu z/OS.

ULW *Pisanie programów obsługi wyjścia kanału w systemie UNIX, Linux, and Windows*

Poniższe informacje ułatwiają pisanie programów obsługi wyjścia kanału dla systemów UNIX, Linux, and Windows .

Wykonaj instrukcje opisane w sekcji “Zapisywanie wyjść i instalowalnych usług w systemach UNIX, Linux i Windows” na stronie 1033. Użyj następujących informacji szczegółowych dotyczących wyjścia kanału, w stosownych przypadkach:

Wyjście musi być zapisane w języku C i jest biblioteką DLL w systemie Windows.

Zdefiniuj fikcyjną procedurę MQStart () w wyjściu i określ `MQStart` jako punkt wejścia w bibliotece. Rysunek 111 na stronie 1067 pokazuje, jak skonfigurować wpis do 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 111. Przykładowy kod źródłowy dla wyjścia kanału

Podczas zapisywania wyjść kanału dla produktu Windows przy użyciu programu Visual C + + należy napisać własny plik DEF . Przykład pokazany w programie Rysunek 112 na stronie 1067. Więcej informacji na temat pisania programów obsługi wyjścia kanału znajduje się w sekcji “Pisanie programów obsługi wyjścia kanału” na stronie 1062.

```
EXPORTS
ChannelExit
```

Rysunek 112. Przykładowy plik DEF dla Windows

Pojęcia pokrewne

“Pisanie programów obsługi wyjścia kanału w systemie IBM i” na stronie 1066
Poniższe informacje umożliwiają pisanie i kompilowanie programów obsługi wyjścia kanału dla produktu IBM i.

“Pisanie programów obsługi wyjścia kanału w systemie z/OS” na stronie 1064
Poniższe informacje umożliwiają pisanie i kompilowanie programów obsługi wyjścia kanału dla produktu 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 znane jako uwierzytelnianie. Aby określić, że kanał musi używać wyjścia bezpieczeństwa, należy określić nazwę wyjścia w polu SCYEXIT definicji kanału.

Uwaga: Uwierzytelnianie można również uzyskać za pomocą rekordów uwierzytelniania kanału. Rekordy uwierzytelniania kanału zapewniają dużą elastyczność w zapobieganiu dostępowi do menedżerów kolejek od określonych użytkowników i kanałów oraz w odwzorowywanie zdalnych użytkowników na identyfikatory użytkowników produktu IBM MQ . Obsługa protokołu TLS jest również udostępniana przez produkt IBM MQ w celu uwierzytelniania użytkowników i zapewnienia szyfrowania i integralności danych w celu sprawdzenia danych. Więcej informacji na temat protokołu TLS można znaleźć w sekcji Protokoły zabezpieczeń TLS w produkcie IBM MQ. Jeśli jednak nadal wymagane jest bardziej wyrafinowane (lub różne) formy przetwarzania zabezpieczeń, a także inne typy kontroli i kontekstu zabezpieczeń, należy rozważyć zapisanie wyjść zabezpieczeń.

W przypadku wyjść zabezpieczeń zapisanych przed produktem IBM WebSphere MQ 7.1 warto zauważyć, że wcześniejsze wersje produktu IBM MQ odpytywały bazowego dostawcę bezpiecznych gniazd (np. GSKit) w celu określenia nazwy wyróżniającej (SSLPEER) i wystawcy certyfikatu partnera zdalnego (SSLCERTI). W produkcie IBM WebSphere MQ 7.1 dodano obsługę dla zakresu nowych atrybutów zabezpieczeń. W celu uzyskania dostępu do tych atrybutów program IBM WebSphere MQ 7.1 uzyskuje kodowanie DER certyfikatu i używa go do określenia nazwy wyróżniającej podmiotu i wystawcy. Atrybuty nazwy wyróżniającej 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, jak również dane przekazywane do wyjść zabezpieczeń kanału na liście, jak pokazano poniżej:

- MQCD SSLPeerNamePtr
- MQCXP SSLRemCertIssNamePtr

W programie IBM WebSphere MQ 7.1atrybut SERIALNUMBER jest również dołączany do nazwy wyróżniającej podmiotu i zawiera numer seryjny certyfikatu partnera zdalnego. Niektóre atrybuty nazwy wyróżniającej są zwracane w innej kolejności niż w poprzednich wersjach. W rezultacie skład pól SSLPEER i SSLCERTI jest zmieniany w produkcie IBM WebSphere MQ 7.1 z poprzednich wersji i dlatego zaleca się, aby wszystkie wyjścia zabezpieczeń lub aplikacje zależne od tych pól były badane i aktualizowane.

Istniejące filtry nazw węzłów sieci IBM MQ określone za pomocą pola SSLPEER definicji kanału nie będą miały wpływu i działają w taki sam sposób, jak we wcześniejszych wersjach. Wynika to z faktu, że algorytm uzgadniania nazw węzłów sieci IBM MQ został zaktualizowany tak, aby przetwarzać istniejące filtry SSLPEER bez konieczności zmiany definicji kanału. Zmiana ta najprawdopodobniej ma wpływ na wyjścia zabezpieczeń i aplikacje zależne od wartości nazwy wyróżniającej podmiotu i nazwy wyróżniającej wystawcy zwracanych przez interfejs programistyczny PCF.

Wyjście zabezpieczeń może być zapisane w języku C lub Java.

Programy obsługi wyjścia zabezpieczeń kanału są wywoływane w następujących miejscach w cyklu przetwarzania agenta MCA:

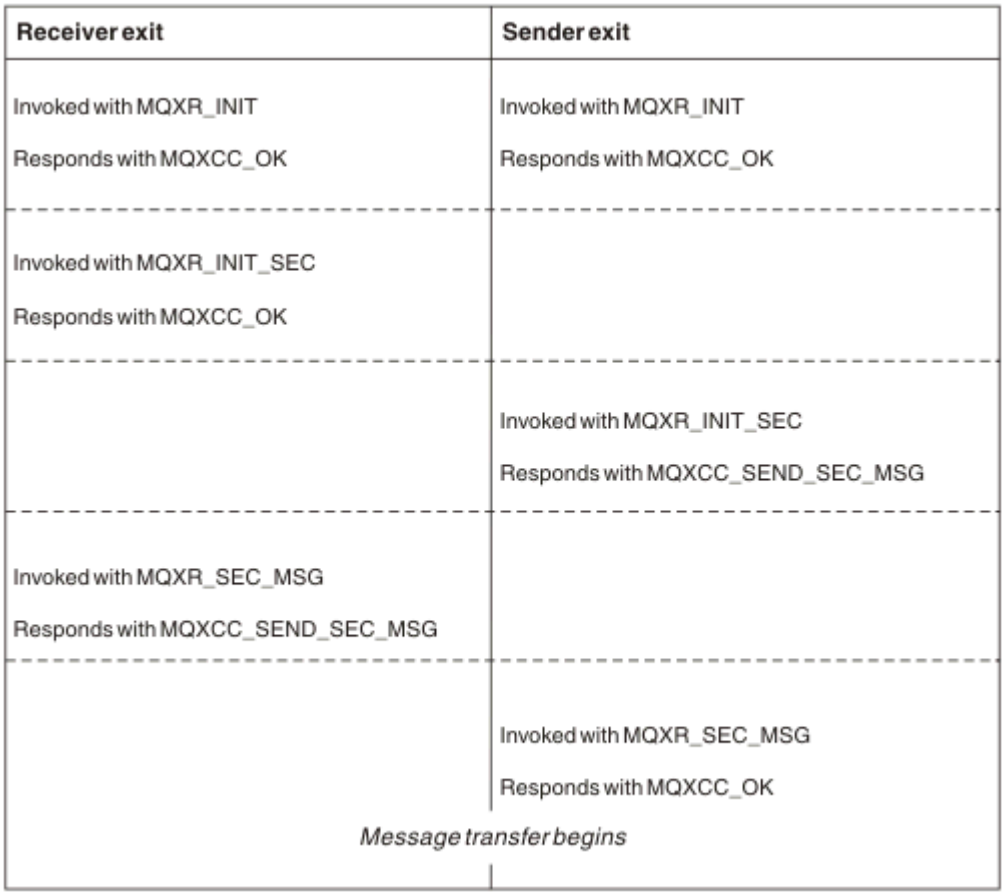
- W momencie inicjowania i zakończenia MCA.
- Natychmiast po zakończeniu początkowego negocjowania danych przy uruchamianiu kanału. Odbiornik lub serwer końca kanału może inicjować wymianę komunikatów zabezpieczeń ze zdalnym końcem, udostępniając komunikat, który ma zostać dostarczony do wyjścia zabezpieczeń na zdalnym końcu. Może to również pogorszać się w tym zakresie. Program obsługi wyjścia jest ponownie uruchamiany w celu przetworzenia dowolnego komunikatu zabezpieczeń odebranego ze zdalnego zakończenia.
- Natychmiast po zakończeniu początkowego negocjowania danych przy uruchamianiu kanału. Koniec kanału wysyłającego lub requestera przetwarza komunikat bezpieczeństwa odebrany ze zdalnego punktu końcowego lub inicjuje wymianę zabezpieczeń, gdy zdalny koniec nie może zostać. Program obsługi wyjścia jest ponownie uruchamiany w celu przetworzenia wszystkich kolejnych komunikatów bezpieczeństwa, które mogą zostać odebrane.

Kanał requestera nigdy nie jest wywoływany za pomocą wywołania MQXR_INIT_SEC. Kanał powiadamia serwer o tym, że ma program obsługi wyjścia zabezpieczeń, a następnie serwer ma możliwość zainicjowania wyjścia zabezpieczeń. Jeśli nie ma on jednego, informuje o tym requester, a przepływ o zerowej długości jest zwracany do programu obsługi wyjścia.

Uwaga: Należy unikać wysyłania komunikatów bezpieczeństwa o zerowej długości.

Przykłady danych wymienianych przez programy obsługi wyjścia zabezpieczeń są ilustrowane w rysunkach Rysunek 113 na stronie 1069 za pomocą Rysunek 116 na stronie 1071. W tych przykładach przedstawiono sekwencję zdarzeń, które występują w przypadku wyjścia zabezpieczeń odbiornika, a także wyjście zabezpieczeń nadawcy. Kolejne wiersze w liczbach przedstawiają upływ czasu. W niektórych przypadkach zdarzenia u odbiorcy i nadawcy nie są skorelowane, a więc mogą wystąpić w tym samym czasie lub w różnych momentach. W innych przypadkach zdarzenie w jednym programie obsługi wyjścia skutkuje zdarzeniem uzupełniającym występującym później w drugim programie obsługi wyjścia. Na przykład w produkcie Rysunek 113 na stronie 1069:

1. Odbiornik i nadawca są wywoływane za pomocą wywołania MQXR_INIT, ale te wywołania nie są skorelowane i mogą być wykonywane w tym samym czasie lub w różnych momentach.
2. Odbiornik jest wywoływany przy użyciu wywołania MQXR_INIT_SEC, ale zwraca MQXCC_OK, które nie wymaga żadnych dodatkowych zdarzeń przy wyjściu nadawcy.
3. Nadawca jest wywoływany przy użyciu wywołania MQXR_INIT_SEC. Nie jest to skorelowane z wywołaniem odbiornika z MQXR_INIT_SEC. Nadawca zwraca wartość MQXCC_SEND_SEC_MSG, co powoduje, że zdarzenie uzupełniające jest w wyjściu odbiornika.
4. Odbiornik jest następnie wywoływany za pomocą MQXR_SEC_MSG i zwraca MQXCC_SEND_SEC_MSG, co powoduje, że zdarzenie uzupełniające jest wykonywane przy wyjściu nadawcy.
5. Nadawca jest następnie wywoływany za pomocą MQXR_SEC_MSG i zwraca MQXCC_OK, które nie wymaga żadnego dodatkowego zdarzenia w wyjściu odbiornika.



Rysunek 113. Wymiana zainicjowana przez nadawcę z umową

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_OK	
	Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_OK	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_SUPPRESS_FUNCTION
Channel closes	
Invoked with MQXR_TERM Responds with MQXCC_OK	Invoked with MQXR_TERM Responds with MQXCC_OK

Rysunek 114. Wymiana zainicjowana przez nadawcę bez umowy

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_SEND_SEC_MSG
Invoked with MQXR_SEC_MSG Responds with MQXCC_OK	
Message transfer begins	
Invoked with MQXR_TERM Responds with MQXCC_OK	Invoked with MQXR_TERM Responds with MQXCC_OK

Rysunek 115. Wymiana inicjowana przez odbiorcę z umową

Receiver exit	Sender exit
Invoked with MQXR_INIT Responds with MQXCC_OK	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_SEND_SEC_MSG	
	Invoked with MQXR_SEC_MSG Responds with MQXCC_OK
Invoked with MQXR_SEC_MSG Responds with MQXCC_SUPPRESS_FUNCTION	
Channel closes	

Rysunek 116. Wymiana inicjowana przez odbiorcę bez zgody

Program obsługi wyjścia zabezpieczeń kanału jest przekazywany do buforu agenta zawierającego dane bezpieczeństwa, z wyjątkiem wszystkich nagłówków transmisji, generowanych przez wyjście zabezpieczeń. Dane te mogą być dowolnymi danymi, dzięki czemu albo zakończenie kanału będzie w stanie przeprowadzić sprawdzanie poprawności zabezpieczeń.

Program obsługi wyjścia zabezpieczeń zarówno w wysyłającym, jak i odbierającym końcu kanału komunikatów może zwrócić jeden z dwóch kodów odpowiedzi do dowolnego wywołania:

- Wymiana zabezpieczeń zakończyła się bez błędów
- Pomijaj kanał i zamknij

Uwaga:

1. Wyjścia zabezpieczeń kanału zwykle działają w parach. Po zdefiniowaniu odpowiednich kanałów należy się upewnić, że zgodne programy obsługi wyjścia są nazwane dla obu końców kanału.
2. IBM i W programie IBM i programy obsługi wyjścia zabezpieczeń, które zostały skompilowane za pomocą komendy `Use adopted authority` (USEADPAUT = *YES), mogą adoptować uprawnienia QMQM lub QMQMADM. Należy zwrócić uwagę, że wyjście nie korzysta z tej funkcji, aby utworzyć zagrożenie bezpieczeństwa systemu.
3. W kanale TLS, na którym drugi koniec kanału udostępnia certyfikat, wyjście zabezpieczeń odbiera nazwę wyróżniającą tematu tego certyfikatu w polu MQCD, do którego dostęp jest uzyskiwany przez parametr SSLPeerNamePtr oraz nazwę wyróżniającą wystawcy w polu MQCXP, do którego dostęp jest uzyskiwany za pomocą komendy SSLRemCertIssNamePtr. Użycie, do którego można umieścić tę nazwę, to:
   - Aby ograniczyć dostęp do kanału TLS.
   - Aby zmienić MQCD.MCAUserIdentifier oparty na nazwie.

Pojęcia pokrewne

Rekordy uwierzytelniania kanału

Pojęcia związane z protokołem TLS (Transport Layer Security)

Zapisywanie wyjścia zabezpieczeń

Wyjście zabezpieczeń można zapisać, korzystając z kodu szkieletu wyjścia zabezpieczeń.

Rysunek 117 na stronie 1072 ilustruje sposób pisania 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 117. Kod szkieletu wyjścia zabezpieczeń

Musi istnieć standardowy punkt wejścia MQStart produktu IBM MQ , ale nie jest on wymagany do wykonywania żadnej funkcji. Nazwa funkcji (w tym przykładzieEntryPoint) może zostać zmieniona, ale funkcja musi zostać wyeksportowana, gdy biblioteka jest kompilowana i dowiązana. Podobnie jak w poprzednim przykładzie, wskaźniki pChannelExitParms muszą być rzutowane na wartość PMQCXP, a definicja pChannelmusi być rzutowana na wartość PMQCD. Ogólne informacje na temat wywoływania wyjść kanału oraz korzystania z parametrów zawiera sekcja MQ_CHANNEL_EXIT. Parametry te są używane w wyjściu zabezpieczeń w następujący sposób:

PMQVOID pChannelExitParms
Wejście/wyjście

Wskaźnik do struktury MQCXP-rzutowanie na wartość PMQCXP w celu uzyskania dostępu do pól. Ta struktura jest używana do komunikowania się między wyjściem a MCA. Następujące pola w MQCXP są szczególnie interesujące dla Exits Security:

ExitReason
Informuje o tym, że program Security Exit jest bieżącym stanem w wymianie zabezpieczeń i jest używany podczas podejmowania decyzji o działaniu, które ma zostać wykonane.

ExitResponse
Odpowiedź na MCA, która dyktuje kolejny etap w wymianie bezpieczeństwa.

ExitResponse2
Dodatkowe opcje sterujące umożliwiające zarządzanie sposobem interpretowania odpowiedzi przez agenta MCA w odpowiedzi na wyjście zabezpieczeń.

Obszar ExitUser
16 bajtów (maksimum) pamięci masowej, które mogą być używane 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 PMQVOID pChannel
Wejście/wyjście

Wskaźnik do struktury MQCD-rzutowanie na wartość PMQCD w celu uzyskania dostępu do pól. Ten parametr zawiera definicję kanału. Następujące pola na zmaterializowanych tabelach MQCD są szczególnie interesujące dla Exits Security:

ChannelName
Nazwa kanału (20 bajtów dopełniona z prawej strony znakami odstępu).

ChannelType
Kod definiujący typ kanału.

Identyfikator użytkownika MCA
Ta grupa trzech pól jest inicjowana do wartości pola MCAUSER określonego w definicji kanału. Każdy identyfikator użytkownika określony przez wyjście zabezpieczeń w tych polach jest używany do kontroli dostępu (nie dotyczy kanałów SDR, SVR, CLNTCONN lub CLUSSDR).

MCAUserIdentifier
Pierwsze 12 bajtów identyfikatora dopełnione jest z prawej strony spacjami.

LongMCAUserIdPtr
Wskaźnik do buforu zawierającego identyfikator pełnej długości (brak gwarantowanej wartości NULL został zakończony) ma pierwszeństwo przed identyfikatorem MCAUserIdentifier.

LongMCAUserIdLength
Długość łańcucha wskazywaanego przez LongMCAUserIdPtr -musi być ustawiona, jeśli ustawiona jest wartość LongMCAUserIdPtr .

Identyfikator użytkownika zdalnego
Ma zastosowanie tylko do par kanałów CLNTCONN/SVRCONN. Jeśli nie zdefiniowano wyjścia zabezpieczeń CLNTCONN, to te trzy pola są inicjowane przez agenta MCA klienta, tak więc mogą zawierać identyfikator użytkownika ze środowiska klienta, który może być używany przez wyjście zabezpieczeń SVRCONN do uwierzytelniania oraz podczas określania identyfikatora użytkownika MCA. Jeśli zdefiniowano wyjście zabezpieczeń CLNTCONN, to pola te nie są inicjowane i można je ustawić za pomocą komendy CLNTCONN Security Exit (Wyjście zabezpieczeń CLNTCONN) lub komunikaty zabezpieczeń mogą być używane do przekazywania identyfikatora użytkownika z klienta do serwera.

Identyfikator RemoteUser
Pierwsze 12 bajtów identyfikatora dopełnione z prawej strony spacjami.

LongRemoteUserIdPtr
Wskaźnik do buforu zawierającego identyfikator pełnej długości (brak gwarantowanej wartości NULL został zakończony) ma wyższy priorytet niż identyfikator RemoteUser.

LongRemoteDługośćUserId
Długość łańcucha wskazanego przez parametr LongRemoteUserIdPtr-musi być ustawiona, jeśli ustawiona jest wartość LongRemoteUserIdPtr.

Długość PMQLONG pData
Wejście/wyjście

Wskaźnik do tabeli MQLONG. Zawiera długość dowolnego wyjścia zabezpieczeń zawartego w polu AgentBuffer po wywołaniu procedury zewnętrznej zabezpieczeń. Musi być ustawiony przez wyjście zabezpieczeń na długość dowolnego komunikatu wysyłanego w AgentBuffer lub ExitBuffer.

PMQLONG pAgentBufferLength
wejściowe,

Wskaźnik do tabeli MQLONG. Długość danych zawartych w AgentBuffer przy wywoływaniu wyjścia zabezpieczeń.

Bufor PMQVOID pAgent
Wejście/wyjście

W przypadku wywołania wyjścia zabezpieczeń wskazuje to na dowolny komunikat wysłany z wyjścia partnera. Jeśli parametr ExitResponse2 w strukturze MQCXP ma ustawioną flagę MQXR2_USE_AGENT_BUFFER (wartość domyślna), to wyjście zabezpieczeń musi ustawić ten parametr w taki sposób, aby wskazywało na wysyłane dane komunikatu.

PMQLONG pExitBufferLength
Wejście/wyjście

Wskaźnik do tabeli MQLONG. Ten parametr jest inicjowany na 0 przy pierwszym wywołaniu procedury zewnętrznej zabezpieczeń, a zwracana wartość jest utrzymywana między wywołaniami wyjścia zabezpieczeń podczas wymiany zabezpieczeń.

PMQPTR pExitBufferAddr
Wejście/wyjście

Ten parametr jest inicjowany do pustego wskaźnika przy pierwszym wywołaniu procedury zewnętrznej zabezpieczeń, a zwracana wartość jest utrzymywana między wywołaniami wyjścia zabezpieczeń podczas wymiany zabezpieczeń. Jeśli opcja MQXR2_USE_EXIT_BUFFER jest ustawiona w elemencie ExitResponse2 w strukturze MQCXP, to wyjście zabezpieczeń musi ustawić ten parametr w taki sposób, aby wskazywało na wysyłane dane komunikatu.

Różnice w zachowaniu między wyjściami bezpieczeństwa zdefiniowanymi w parach kanału CLNTCONN/SVRCONN i innymi parami kanału
Wyjścia bezpieczeństwa mogą być definiowane na wszystkich typach kanałów. Jednak zachowanie wyjść zabezpieczeń zdefiniowanych w parach kanału CLNTCONN/SVRCONN różni się nieco od wyjść zabezpieczeń zdefiniowanych w innych parach kanałów.

Wyjście zabezpieczeń na kanale CLNTCONN może ustawić identyfikator zdalnego użytkownika w definicji kanału na potrzeby przetwarzania przez partnerskie wyjście SVRCONN lub dla autoryzacji OAM, jeśli nie zdefiniowano wyjścia zabezpieczeń SVRCONN, a pole MCAUSER parametru SVRCONN nie jest ustawione.

Jeśli nie zdefiniowano wyjścia zabezpieczeń CLNTCONN, to identyfikator zdalnego użytkownika w definicji kanału jest ustawiany na identyfikator użytkownika ze środowiska klienta (który może być pusty) przez agenta MCA klienta.

Wymiana zabezpieczeń między wymianą zabezpieczeń zdefiniowana w parze kanałów CLNTCONN i SVRCONN zakończy się pomyślnie, gdy wyjście zabezpieczeń SVRCONN zwróci wartość ExitResponse w przypadku wywołania MQXCC_OK. Wymiana zabezpieczeń między innymi parami kanału zakończy się pomyślnie, gdy wyjście zabezpieczeń, które zainicjował wymianę, zwróci wartość ExitResponse (MQXCC_OK).

Jednak kod MQXCC_SEND_AND_REQUEST_SEC_MSG ExitResponse może być używany w celu wymuszenia kontynuacji wymiany zabezpieczeń: jeśli ExitResponse z MQXCC_SEND_AND_REQUEST_SEC_MSG jest zwracana przez komendę CLNTCONN lub SVRCONN Security Exit, wyjście partnerskie musi odpowiedzieć, wysyłając komunikat bezpieczeństwa (a nie MQXCC_OK lub odpowiedź null) lub kanał kończy działanie. W przypadku wyjść zabezpieczeń zdefiniowanych dla innych typów kanału odpowiedź ExitResponse MQXCC_OK zwrócona w odpowiedzi na wartość MQXCC_SEND_AND_REQUEST_SEC_MSG z partnerskiego wyjścia zabezpieczeń powoduje kontynuację wymiany zabezpieczeń tak, jakby została zwrócona odpowiedź o wartości NULL, a nie podczas kończenia działania kanału.

Wyjście zabezpieczeń SSPI
Produkt IBM MQ for Windows dostarcza wyjście zabezpieczeń, które udostępnia uwierzytelnianie dla kanałów produktu IBM MQ przy użyciu interfejsu SSPI (Security Services Programming Interface). Interfejs SSPI udostępnia zintegrowane zabezpieczenia produktu Windows.

To wyjście zabezpieczeń jest przeznaczone zarówno dla klienta IBM MQ , jak i dla serwera IBM MQ .

Pakiety zabezpieczeń są ładowane z pliku security.dll lub z pliku secur32.dll. Te biblioteki DLL są dostarczane wraz z systemem operacyjnym.

Uwierzytelnianie jednokierunkowe jest udostępniane w produkcie Windowsprzy użyciu usług uwierzytelniania NTLM. Uwierzytelnianie dwukierunkowe jest udostępniane w produkcie Windows 2000przy użyciu usług uwierzytelniania Kerberos .

Program obsługi wyjścia zabezpieczeń jest dostarczany w formacie źródłowym i obiektowym. Można użyć kodu wynikowego, który jest, lub użyć kodu źródłowego jako punktu wyjścia do tworzenia własnych programów obsługi wyjścia użytkownika. Więcej informacji na temat korzystania z obiektu lub kodu źródłowego wyjścia zabezpieczeń SSPI zawiera sekcja “Korzystanie z wyjścia zabezpieczeń SSPI w systemie Windows” na stronie 1246

Programy obsługi wyjścia wysyłania i odbierania kanału
Istnieje możliwość użycia wyjść wysyłania i odbierania w celu wykonania takich zadań, jak kompresja danych i dekompresja danych. Istnieje możliwość określenia listy programów obsługi wyjścia wysyłania i odbierania, które mają być uruchamiane w ramach dziedziczenia.

Programy obsługi wyjścia wysyłania i odbierania kanału są wywoływane w następujących miejscach w cyklu przetwarzania agenta MCA:

- Programy obsługi wyjścia wysyłania i odbierania są wywoływane do inicjowania w inicjacji MCA, a także do zakończenia w momencie zakończenia agenta MCA.
- Program obsługi wyjścia wysyłania jest wywoływany na jednym lub drugim końcu kanału, w zależności od zakończenia transmisji dla jednego przesyłania komunikatów, bezpośrednio przed wysłaniem transmisji przez łącze. Uwaga 4 wyjaśnia, dlaczego wyjścia są dostępne w obu kierunkach, mimo że kanały komunikatów wysyłają wiadomości tylko w jednym kierunku.
- Program obsługi wyjścia odbierania jest wywoływany na jednym lub drugim końcu kanału, w zależności od zakończenia transmisji dla jednego przesyłania komunikatów, natychmiast po odebraniu transmisji z łącza. Uwaga 4 wyjaśnia, dlaczego wyjścia są dostępne w obu kierunkach, mimo że kanały komunikatów wysyłają wiadomości tylko w jednym kierunku.

Może istnieć wiele transmisji dla jednego transferu komunikatów, a może istnieć wiele iteracji programów obsługi wyjścia wysyłania i odbierania, zanim komunikat osiągnie wyjście komunikatu na końcu odbierającym.

Programy obsługi wyjścia wysyłania i odbierania kanału są przekazywane do buforu agenta zawierającego dane transmisji jako wysłane lub odebrane z łącza komunikacyjnego. W przypadku programów obsługi wyjścia wysyłania pierwszych 8 bajtów buforu jest zarezerwowanych do użycia przez agenta MCA i nie może być zmieniane. Jeśli program zwraca inny bufor, to te pierwsze 8 bajtów musi znajdować się 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 (abend).

Uwaga: Nie należy wywoływać wywołania MQGET, MQPUT lub MQPUT1 w punkcie synchronizacji z poziomu wyjścia wysyłania lub odbierania.

Uwaga:

1. Wyjścia wysyłania i odbierania zwykle pracują w parach. Na przykład wyjście wysyłania może kompresować dane, a wyjście odbierania jest dekompresowane, albo wyjście wysyłania może szyfrować dane, a wyjście odbierania odszyfrować je. Po zdefiniowaniu odpowiednich kanałów należy się upewnić, że zgodne programy obsługi wyjścia są nazwane dla obu końców kanału.
2. Jeśli kompresja jest włączona dla kanału, wyjścia są przekazywane skompresowane dane.
3. Programy zewnętrzne wysyłania i odbierania kanału mogą być wywoływane dla segmentów komunikatów innych niż dla danych aplikacji, na przykład komunikatów o statusie. Nie są one wywoływane podczas uruchamiania okna dialogowego ani fazy sprawdzania zabezpieczeń.
4. Mimo że kanały komunikatów wysyłają komunikaty tylko w jednym kierunku, dane sterowania kanałami, takie jak bity serca i koniec przetwarzania wsadowego, przepływają w obu kierunkach, a wyjścia te są dostępne w obu kierunkach, również. Jednak niektóre z początkowych przepływów danych uruchamiania kanału są zwolnione z przetwarzania przez żaden z wyjść.
5. Istnieją okoliczności, w których można wywołać wyjścia wysyłania i odbierania z sekwencji, na przykład w przypadku uruchamiania serii programów obsługi wyjścia lub w przypadku uruchamiania wyjść zabezpieczeń. Następnie, po pierwszym wywołaniu wyjścia odbierania w celu przetwarzania danych, może on odbierać dane, które nie zostały przekazane przez odpowiednie wyjście wysyłania. Jeśli program obsługi wyjścia odbierania wykonał operację, na przykład dekompresję, nie sprawdzając przy tym, czy jest ona wymagana, wyniki byłyby nieoczekiwane.

   Należy zakodować wyjścia wysyłania i odbierania w taki sposób, aby wyjście odbierania było możliwe sprawdzenie, czy dane, które jest odbierany, zostały przetworzone przez odpowiednie wyjście wysyłania. Zalecanym sposobem działania jest zakodowanie programów obsługi wyjścia w taki sposób, aby:

   - Wyjście wysyłania ustawia wartość dziewiątego bajtu danych na 0 i przenosi wszystkie dane wzdłuż 1 bajtu, przed wykonaniem operacji. (Pierwsze 8 bajtów jest zarezerwowane do użycia przez agenta MCA).
   - Jeśli wyjście odbierania odbiera dane, które mają wartość 0 w bajcie 9, to wie, że dane pochodzą z wyjścia wysyłania. Usuwa wartość 0, wykonuje operację komplementarną i przenosi dane wynikowe z powrotem o 1 bajt.
   - Jeśli wyjście odbierania odbiera dane, które mają coś innego niż 0 w bajcie 9, zakłada, że wyjście wysyłania nie zostało uruchomione, a następnie wysyła dane z powrotem do programu wywołującego w niezmienionej postaci.

   W przypadku korzystania z wyjść zabezpieczeń, jeśli kanał jest zakończony przez wyjście zabezpieczeń, możliwe jest wywołanie wyjścia wysyłania bez odpowiadającego mu wyjścia odbierania. Jednym ze sposobów zapobiegania temu problemowi jest zakodowanie wyjścia zabezpieczeń w celu ustawienia flagi, na przykład MQCD.SecurityUserData lub MQCD.SendUserData, gdy wyjście decyduje o zakończeniu działania kanału. Następnie wyjście wysyłania musi sprawdzić to pole i przetwarzać dane tylko wtedy, gdy flaga nie jest ustawiona. To sprawdzenie zapobiega niepotrzebnej zmianie danych przez wyjście wysyłania, a tym samym zapobiega występowaniu błędów konwersji, które mogą wystąpić, jeśli wyjście zabezpieczeń odebrało zmienione dane.

Programy obsługi wyjścia wysyłania kanału-zmiana miejsca na dysku
Istnieje możliwość użycia wyjść wysyłania i odbierania w celu transformowania danych przed transmisją. Programy obsługi wyjścia wysyłania kanału mogą dodawać własne dane na temat transformacji przez ponowne udostępnianie miejsca 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 zarezerwować miejsce i korzystać z niego

Gdy program obsługi wyjścia wysyłania jest wywoływany w celu zainicjowania, ustaw pole *ExitSpace* zmaterializowanej tabeli zapytania (MQXCP) na liczbę bajtów, które mają być zarezerwowane. Szczegółowe informacje na ten temat zawiera sekcja MQCXP . Parametr *ExitSpace* można ustawić tylko podczas inicjowania, to znaczy, gdy wartość *ExitReason* ma wartość MQXR_INIT. Gdy wyjście wysyłania jest wywoływane bezpośrednio przed transmisją, z *ExitReason* ustawionym na MQXR_XMIT, bajty *ExitSpace* są zarezerwowane w buforze transmisji. Produkt *ExitSpace* nie jest obsługiwany w systemie z/OS.

Wyjście wysyłania nie musi używać całej zarezerwowanej przestrzeni. Może używać mniej niż *ExitSpace* bajtów lub, jeśli bufor transmisji nie jest zapełniony, wyjście może wykorzystać więcej niż zarezerwowaną kwotę. Podczas ustawiania wartości parametru *ExitSpace*należy pozostawić co najmniej 1 kB dla danych komunikatu w buforze transmisji. Wydajność kanałów może mieć wpływ, jeśli zarezerwowane miejsce jest używane dla dużych ilości danych.

Bufor transmisji ma zwykle długość 32KB . Jeśli jednak kanał używa protokołu TLS, wielkość buforu transmisji zostanie zmniejszona do 15,352 bajtów w celu dopasowania do maksymalnej długości rekordu zdefiniowanej w dokumencie RFC 6101 i pokrewnej rodzinie standardów TLS. Dalsze 1024 bajty są zarezerwowane do użycia przez produkt IBM MQ, więc maksymalna przestrzeń buforu transmisji użyteczna przez wyjścia nadawcze wynosi 14,328 bajtów.

Co dzieje się na odbierającym końcu kanału

Programy obsługi wyjścia odbierania kanału muszą być skonfigurowane w taki sposób, aby były zgodne z odpowiednimi wyjściami nadawczym. Wyjścia odbierania muszą znać liczbę bajtów w zarezerwowanym obszarze i muszą usunąć te dane w tym obszarze.

Wiele wyjść wysyłania

Istnieje możliwość określenia listy programów obsługi wyjścia wysyłania i odbierania, które mają być uruchamiane w ramach dziedziczenia. Produkt IBM MQ przechowuje sumę dla obszaru zarezerwowanego przez wszystkie wyjścia wysyłania. W tym obszarze musi być co najmniej 1 kB dla danych komunikatu w buforze transmisji.

W poniższym przykładzie pokazano, w jaki sposób przestrzeń jest przydzielana dla trzech wyjść nadawanych, wywołanych kolejno:

1. Po wywołaniu do inicjalizacji:
   - Wysłanie wyjścia A rezerwuje 1 KB.
   - Wysłanie wyjścia B rezerwuje 2 KB.
   - Wysłanie wyjścia C rezerwuje 3 KB.
2. Maksymalna wielkość transmisji wynosi 32 kB, a dane użytkownika o długości 5 kB.
3. Wyjście A jest wywoływane z 5 KB danych; do 27 KB są dostępne, ponieważ 5 KB jest zarezerwowane dla wyjść B i C. Wyjście A dodaje 1 kB, ilość zarezerwowana.
4. Wyjście B jest wywoływane z 6 KB danych; dostępne są do 29 KB, ponieważ 3 KB jest zarezerwowane dla wyjścia C. Wyjście B dodaje 1 KB, mniej niż 2 KB zarezerwowanego.
5. Wyjście C jest wywoływane z 7 KB danych; do 32 kB dostępne są dane. Wyjście C dodaje 10K, więcej niż zarezerwowane 3 KB. Ta kwota jest poprawna, ponieważ łączna ilość danych, 17 kB, jest mniejsza niż maksimum 32 kB.

Maksymalna wielkość buforu transmisji dla kanału używała protokołu TLS to 15,352 bajtów, a nie 32KB. Wynika to z faktu, że podstawowe segmenty bezpiecznej transmisji gniazda są ograniczone do 16KB , a część przestrzeni jest wymagana dla rekordów protokołu TLS. Dalsze 1024 bajty są zarezerwowane do użycia przez produkt IBM MQ, więc maksymalna przestrzeń buforu transmisji użyteczna przez wyjścia nadawcze wynosi 14,328 bajtów.

Programy obsługi wyjścia komunikatów kanału
Wyjścia komunikatów kanału można użyć do wykonywania zadań, takich jak szyfrowanie łącza, sprawdzanie poprawności lub zastępowanie przychodzących identyfikatorów użytkowników, konwersja danych komunikatów, kronikowanie i obsługa komunikatów referencyjnych. Istnieje możliwość określenia listy programów obsługi wyjścia komunikatów, które mają być uruchamiane w ramach dziedziczenia.

Programy obsługi wyjścia komunikatów kanału są wywoływane w następujących miejscach w cyklu przetwarzania agenta MCA:

- Inicjacja i zakończenie MCA
- Natychmiast po wysłaniu przez wysyłającego agenta MCA wywołania MQGET
- Zanim odbierający agent MCA zgłosi wywołanie MQPUT

Wyjście komunikatu jest przekazywane do buforu agenta zawierającego nagłówek kolejki transmisji MQXQH, a także tekst komunikatu aplikacji pobrany z kolejki. Format MQXQH jest podany w składce MQXQH-nagłówek kolejki transmisji.

Jeśli używane są komunikaty odniesienia (to znaczy komunikaty, które zawierają tylko nagłówek wskazujący inny obiekt, który ma zostać wysłany), wyjście komunikatu rozpoznaje nagłówek, MQRMH. Identyfikuje obiekt, pobiera go w dowolny sposób, dodaje go do nagłówka i przekazuje je do agenta MCA w celu przesłania do odbierającego agenta MCA. W odbierającym MCA inny program obsługi wyjścia rozpoznaje, że ten komunikat jest komunikatem odniesienia, wyodrębnia obiekt i przekazuje nagłówek do kolejki docelowej. Więcej informacji na temat komunikatów referencyjnych oraz niektórych przykładowych wyjść komunikatów, które obsługują te komunikaty, można znaleźć w sekcji “Komunikaty odniesienia” na stronie 887 i “Uruchamianie przykładów komunikatów odniesienia” na stronie 1215 .

Wyjścia komunikatów mogą zwracać następujące odpowiedzi:

- Wyślij wiadomość (wyjście GET). Komunikat mógł zostać zmieniony przez wyjście. (Ta wartość zwraca wartość MQXCC_OK).
- Umieść komunikat w kolejce (wyjście PUT). Komunikat mógł zostać zmieniony przez wyjście. (Ta wartość zwraca wartość MQXCC_OK).
- Nie przetwarzaj komunikatu. Komunikat jest umieszczany w kolejce niedostarczonych komunikatów (niedostarczona kolejka komunikatów) przez agenta MCA.
- Zamknij kanał.
- Błędny kod powrotu, który powoduje nieprawidłowe zakończenie działania agenta MCA.

Uwaga:

1. Wyjścia komunikatów są wywoływane raz dla każdego przesyłanego kompletnego komunikatu, nawet jeśli komunikat jest podzielony na części.
2. Linux UNIX Jeśli zostanie wyświetlony komunikat wyjścia komunikatów w systemie UNIX lub Linux, automatyczna konwersja identyfikatorów użytkowników na małe litery (opisane tutaj) nie działa.
3. Wyjście jest uruchamiane w tym samym wątku co sam agent MCA. Działa on również wewnątrz tej samej jednostki pracy (UOW) co agent MCA, ponieważ korzysta z tego samego uchwytu połączenia. Tak więc wszystkie wywołania wykonane w punkcie synchronizacji są zatwierdzane lub wycofane 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 tylko w kolejce, gdy zadanie wsadowe zawierające oryginalny komunikat zostanie zatwierdzone.

   Z tego powodu możliwe jest wydanie wywołań MQI punktów synchronizacji z programu obsługi wyjścia komunikatów kanału.

Konwersja komunikatów poza wyjściem komunikatu
Przed wywołaniem wyjścia komunikatu odbierający agent MCA wykonuje pewne konwersje w komunikacie. W tej sekcji opisano algorytmy używane do przeprowadzania 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 przez połączone nagłówki, które są zgodne z MQXQH, w razie potrzeby przeprowadzając konwersję. Nagłówki łańcuchowe mogą wykraczać poza przesunięcie zawarte w parametrze HeaderLength danych MQCXP przekazywanych do wyjścia komunikatu odbiornika. Następujące nagłówki są przekształcane w miejsce:

- MQXQH (nazwa formatu " `MQXMIT` ")
- MQMD (ten nagłówek jest częścią tabeli 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 one stopniowane, ponieważ agent MCA kontynuuje przetwarzanie połączonych nagłówków:

- MQDLH (nazwa formatu " `MQDEAD` ")
- wszystkie nagłówki o nazwach rozpoczynających się od trzech znaków 'MQH' (na przykład " `MQHRF` ") które nie są wymienione w inny sposób

Sposób przetwarzania nagłówków

Parametr Format każdego nagłówka IBM MQ jest odczytyany przez agenta MCA. Parametr Format to 8 bajtów w nagłówku, które są 8 jednobajtowymi znakami zawierającymi nazwę.

Następnie agent MCA interpretuje dane po każdym nagłówku jako typ nazwanego typu. Jeśli format jest nazwą typu nagłówka kwalifikującego się do konwersji danych produktu IBM MQ , jest on przekształcany. Jeśli jest to inna nazwa wskazująca dane inne niżMQ (na przykład MQFMT_NONE lub MQFMT_STRING), agent MCA zatrzyma przetwarzanie nagłówków.

Co to jest HeaderLengthMQCXP?

Parametr HeaderLength w danych MQCXP dostarczanych do wyjścia komunikatów to łączna długość nagłówków MQXQH (w tym MQMD), MQMDE i MQDH na początku komunikatu. Nagłówki te są łańcuchowe przy użyciu nazw i długości formatu 'Format'.

MQWIH

Nagłówki łańcuchowe mogą wykraczać poza obszar HeaderLength do obszaru danych użytkownika. Nagłówek MQWIH, jeśli jest obecny, jest jednym z tych nagłówków, które pojawiają się poza HeaderLength.

Jeśli istnieje nagłówek MQWIH w nagłówkach łańcuchowych, jest on przekształcany w miejscu przed wywołaniem wyjścia komunikatu odbiorcy.

Program obsługi wyjścia dla ponowienia komunikatu kanału
Wywołanie wyjścia komunikatu kanału jest wywoływane w przypadku, gdy próba otwarcia kolejki docelowej nie powiodła się. Można użyć wyjścia, aby określić, w jakich okolicznościach należy ponowić próbę, ile razy należy ponowić próbę, a także jak często.

To wyjście jest również wywoływane na odbierającym końcu kanału w inicjacji MCA i zakończeniu.

Wyjście komunikatu kanału-wyjście ponowienia jest przekazywane do buforu agenta zawierającego nagłówek kolejki transmisji, MQXQH oraz tekst komunikatu aplikacji pobierany z kolejki. Format wartości MQXQH jest podany w sekcji Przegląd dla MQXQH.

Wyjście jest wywoływane dla wszystkich kodów przyczyny; wyjście określa, dla których kodów przyczyny ma być ponawiane przez agenta MCA, przez ile razy i w jakich odstępach czasu. (Wartość licznika

ponowień komunikatu, gdy kanał został zdefiniowany, jest przekazywany do wyjścia na zmaterializowanej tabeli MQCD, ale wyjście może zignorować tę wartość).

Pole Licznik MsgRetryw produkcie MQCXP jest zwiększane przez agenta MCA przy każdym wywołaniu wyjścia, a wyjście zwraca wartość MQXCC_OK z czasem oczekiwania zawartym w polu MsgRetryw polu Odstęp czasu MQCXP lub MQXCC_SUPPRESS_FUNCTION. Ponowne próby są kontynuowane w nieskończoność do momentu, aż wyjście zwróci MQXCC_SUPPRESS_FUNCTION w polu ExitResponse w MQCXP. Informacje na temat działań podjętych przez agenta MCA dla tych kodów zakończenia zawiera sekcja MQCXP .

Jeśli wszystkie ponowienia nie powiodą się, komunikat zostanie zapisany w kolejce niedostarczonych komunikatów. Jeśli nie jest dostępna żadna kolejka niedostarczonych komunikatów, kanał zostanie zatrzymany.

Jeśli dla kanału nie zostanie zdefiniowane wyjście dla ponowienia komunikatu, a wystąpi błąd, który prawdopodobnie będzie tymczasowy, na przykład MQRC_Q_FULL, agent MCA użyje liczby ponowień komunikatu i przedziały czasu ponowienia komunikatu, gdy kanał został zdefiniowany. Jeśli błąd ma charakter bardziej trwały, a użytkownik nie zdefiniował programu obsługi wyjścia do obsługi tego błędu, komunikat jest zapisywany w kolejce niedostarczonych komunikatów.

Program obsługi wyjścia automatycznej definicji kanału
Wyjście automatyczne definicji kanału może być używane, gdy odebrano żądanie uruchomienia kanału odbiorczego lub kanału połączenia z serwerem, ale nie istnieje definicja dla tego kanału (nie dla IBM MQ for z/OS). Można ją również wywołać na wszystkich platformach dla kanałów wysyłających klastry i kanały odbierające klastry w celu umożliwienia modyfikacji definicji dla instancji kanału.

Wyjście automatycznego definiowania kanału może być wywoływane na wszystkich platformach z wyjątkiem z/OS , gdy odebrano żądanie uruchomienia kanału odbiorczego lub kanału połączenia z serwerem, ale nie istnieje definicja kanału. Można go użyć do zmodyfikowania podanej definicji domyślnej dla automatycznie zdefiniowanego kanału odbiorczego lub kanału połączenia z serwerem, SYSTEM.AUTO.RECEIVERlub SYSTEM.AUTO.SVRCON. Informacje na temat automatycznego tworzenia definicji kanałów można znaleźć w sekcji Przygotowywanie kanałów .

Wyjście automatyczne definicji kanału może być również wywoływane po odebraniu żądania w celu uruchomienia kanału nadawczego klastra. Można go wywołać dla kanałów wysyłających klastry i kanały odbierające klastry w celu umożliwienia modyfikacji definicji dla tej instancji kanału. W takim przypadku wyjście ma również zastosowanie do produktu IBM MQ for z/OS. Częstym zastosowaniem wyjścia z 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 automatycznego definiowania kanału, domyślnym zachowaniem w programie z/OS jest sprawdzenie rozproszonej nazwy wyjścia w postaci *[path]/libraryname(function)* i podjęcie maksymalnie ośmiu znaków funkcji, jeśli istnieje, lub nazwy biblioteki. W systemie z/OSprogram obsługi wyjścia automatycznej definicji kanału musi zmieniać pola adresowane przez 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 automatycznie zdefiniowanymi kanałami.

Podobnie jak w przypadku innych wyjść kanału, lista parametrów jest następująca:

```
MQ_CHANNEL_AUTO_DEF_EXIT (ChannelExitParms, ChannelDefinition)
```

Produkt `ChannelExitParms` jest opisany w sekcji MQCXP. Produkt `ChannelDefinition` jest 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 MQ_CHANNEL_AUTO_DEF_EXIT. Jednak próba zmiany innych pól nie powoduje błędu.

Wyjście automatycznego definiowania kanału zwraca odpowiedź albo MQXCC_OK, albo MQXCC_SUPPRESS_FUNCTION. Jeśli żadna z tych odpowiedzi nie zostanie zwrócona, agent MCA będzie kontynuować przetwarzanie, tak jak gdyby zwrócono MQXCC_SUPPRESS_FUNCTION. Oznacza to, że

automatyczna definicja jest porzucona, nie jest tworzona nowa definicja kanału i nie można uruchomić kanału.

ULW *Kompilowanie programów obsługi wyjścia kanału w systemach Windowsi UNIX and Linux*

Poniższe przykłady ułatwiają kompilowanie programów obsługi wyjścia kanału dla systemów Windowsi UNIX and Linux .

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 UNIX and Linux

Linux UNIX

W tych przykładach `exit` jest nazwą biblioteki, a `ChannelExit` jest nazwą funkcji. W systemie AIX plik eksportu nosi nazwę `exit.exp`. Te nazwy są używane przez definicję kanału do odwołania się do programu obsługi wyjścia przy użyciu formatu opisanego w sekcji Definicja kanału MQCD. Zapoznaj się także z parametrem MSGEXIT komendy DEFINE CHANNEL .

AIX Przykładowe komendy kompilatora i konsolidatora dla wyjść kanału 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 kanału 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 kanału 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
```

Solaris Przykładowe komendy kompilatora i konsolidatora dla wyjść kanału w systemie Solaris:

```
$ cc -xarch=v9 -mt -G -o /var/mqm/exits64/exit exit.c -I/opt/mqm/inc
-R/usr/lib/64 -lsocket -lnsl -ldl
```

Na kliencie można użyć wyjścia 32-bitowego lub 64-bitowego. To wyjście musi być połączone z mqic_r.

AIX W systemie AIXwszystkie funkcje wywoływane przez produkt IBM MQ muszą zostać wyeksportowane. Przykładowy plik eksportu dla tego pliku make:

```
#
!channelExit
MQStart
```

Konfigurowanie wyjść kanału

Aby wywołać wyjście kanału, należy podać nazwę w definicji kanału.

Wyjścia kanału muszą być nazwane w definicji kanału. To nazewnictwo można wykonać po pierwszym zdefiniowaniu kanałów lub później można dodać informacje później, na przykład za pomocą komendy MQSC ALTER CHANNEL. Nazwy wyjść kanału można również nadać w strukturze danych kanału MQCD. Format nazwy wyjścia zależy od używanej platformy IBM MQ . Informacje na ten temat można znaleźć w sekcji Komendy MQCD lub MQSC .

Jeśli definicja kanału nie zawiera nazwy programu użytkownika obsługi wyjścia, program zewnętrzny nie jest wywoływany.

Wyjście automatyczne definicji kanału to właściwość menedżera kolejek, a nie pojedynczego kanału. Aby to wyjście zostało wywołane, musi ono być nazwane w definicji menedżera kolejek. Aby zmienić definicję menedżera kolejek, należy użyć komendy MQSC ALTER QMGR.

Pisanie wyjść konwersji danych

Ta kolekcja tematów zawiera informacje na temat sposobu pisania wyjść konwersji danych.

Uwaga: Nieobsługiwane w produkcie MQSeries dla VSE/ESA.

Podczas wykonania operacji MQPUT aplikacja tworzy deskryptor komunikatu (MQMD) komunikatu. Ponieważ produkt IBM MQ musi być w stanie zrozumieć zawartość deskryptora MQMD niezależnie od platformy, na której jest ona tworzona, jest ona automatycznie przekształcana przez system.

Dane aplikacji nie są jednak przekształcane automatycznie. Jeśli dane znakowe są wymieniane między platformami, w których pola `CodedCharSetId` i `Encoding` różnią się między innymi, na przykład między ASCII a EBCDIC, aplikacja musi zorganizować konwersję tego komunikatu. Konwersja danych aplikacji może być wykonywana przez sam menedżer kolejek lub przez program obsługi wyjścia użytkownika, zwany *wyjściem konwersji danych*. Jeśli dane aplikacji znajdują się w jednym z wbudowanych formatów (takich jak MQFMT_STRING), menedżer kolejek może samodzielnie wykonać konwersję danych, korzystając z jednej z wbudowanych procedur konwersji. Ten temat zawiera informacje na temat narzędzia wyjścia konwersji danych, które produkt IBM MQ udostępnia, gdy dane aplikacji nie są w formacie wbudowanym.

Sterowanie może być przekazywane do wyjścia konwersji danych podczas wywołania MQGET. Pozwala to na uniknięcie konwersji na różne platformy przed dotarciem do miejsca docelowego. Jeśli jednak ostatnim miejscem docelowym jest platforma, która nie obsługuje konwersji danych w tabeli MQGET, należy określić CONVERT (YES) w kanale nadawczym, który wysyła dane do jego miejsca docelowego. Dzięki temu program IBM MQ przekształca dane w czasie transmisji. W takim przypadku wyjście konwersji danych musi znajdować się w systemie, w którym zdefiniowany jest kanał nadawczy.

Wywołanie MQGET jest wysyłane bezpośrednio przez aplikację. Ustaw wartości pól `CodedCharSetId` i `Encoding` w strukturze MQMD na wymagany zestaw znaków i kodowanie. Jeśli aplikacja używa tego samego zestawu znaków i kodowania co menedżer kolejek, należy ustawić wartość `CodedCharSetId` na wartość MQCCSI_Q_MGR, a wartość `Encoding` na MQENC_NATIVE. Po zakończeniu wywołania MQGET pola te mają wartości odpowiednie dla zwracanych danych komunikatu. Wartości te mogą się różnić od wartości wymaganych w przypadku, gdy konwersja nie powiodła się. Aplikacja powinna zresetować te pola do wartości wymaganych przed każdym wywołaniem MQGET.

Warunki wymagane dla wyjścia konwersji danych, które mają zostać wywołane, są definiowane dla wywołania MQGET w MQGET.

Opis parametrów, które są przekazywane do wyjścia konwersji danych oraz szczegółowe informacje o używaniu, zawiera sekcja Konwersja danych dla wywołania MQ_DATA_CONV_EXIT i struktury MQDXP.

Programy przekształcające dane aplikacji między różnymi kodowaniami maszyn i identyfikatorami CCSID muszą być zgodne z interfejsem DCI (IBM MQ data conversion interface).

W przypadku klientów rozsyłania grupowego wyjścia funkcji API i wyjścia konwersji danych muszą być możliwe do uruchomienia po stronie klienta, ponieważ niektóre komunikaty mogą nie przechodzić przez menedżer kolejek. Następujące biblioteki są częścią pakietów klienta, a także pakietów serwera:

Tabela 147. Biblioteki, które znajdują się w pakietach klienta i serwera

System operacyjny	Biblioteki
AIX	32 bity & 64 bit: libmqm.a & libmqm_r.a
IBM i	LIBMQM & LIBMQM_R
Linux	32 bity & 64 bit: libmqm.so & libmqm_r.so
Solaris	32 bity & 64 bit: libmqm.so
Windows	32 bity & 64 bit: mqm.dll & mqm.pdb

Wywoływanie wyjścia konwersji danych

Wyjście konwersji danych to wyjście napisane przez użytkownika, które odbiera sterowanie podczas przetwarzania wywołania MQGET.

Wyjście jest wywoływane, jeśli prawdziwe są następujące stwierdzenia:

- Opcja MQGMO_CONVERT została określona w wywołaniu MQGET.
- Niektóre lub wszystkie dane komunikatu nie znajdują się w żądanym zestawie znaków ani kodowaniu.
- Pole *Format* w strukturze MQMD powiązanej z komunikatem nie ma wartości MQFMT_NONE.
- Wartość *BufferLength* podana w wywołaniu MQGET nie jest równa zero.
- Długość danych komunikatu nie jest równa zero.
- Komunikat zawiera dane, które mają format zdefiniowany przez użytkownika. Format zdefiniowany przez użytkownika może zajmować cały komunikat lub być poprzedzony jednym lub większą liczbę wbudowanych formatów. 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.

  Program obsługi wyjścia napisany przez użytkownika można również wywołać w celu przekształcenia wbudowanego formatu, ale dzieje się tak tylko wtedy, gdy wbudowane procedury konwersji nie mogą pomyślnie przekształcić wbudowanego formatu.

Istnieją inne warunki, które zostały w pełni opisane w uwagach dotyczących użycia wywołania MQ_DATA_CONV_EXIT w MQ_DATA_CONV_EXIT.

Szczegółowe informacje na temat wywołania MQGET zawiera sekcja MQGET . Wyjścia konwersji danych nie mogą używać wywołań MQI, innych niż wywołania MQXCNVC.

Nowa kopia wyjścia jest ładowana, gdy aplikacja próbuje pobrać pierwszy komunikat, który używa tego produktu *Format* od momentu połączenia aplikacji z menedżerem kolejek. Nowa kopia może być również ładowana w innych sytuacjach, jeśli menedżer kolejek odrzuciło wcześniej załadowaną kopię.

Wyjście konwersji danych jest uruchamiane w środowisku takim jak program, który wywołał wywołanie MQGET. Program może także być agentem MCA (agenta kanału komunikatów) wysyłającym komunikaty do docelowego menedżera kolejek, który nie obsługuje konwersji komunikatów. Środowisko obejmuje przestrzeń adresową i profil użytkownika, tam gdzie ma to zastosowanie. Wyjście nie może spowodować naruszenia integralności menedżera kolejek, ponieważ nie jest ono uruchamiane w środowisku menedżera kolejek.

Konwersja danych w systemie z/OS

z/OS

W systemie z/OSnależy pamiętać o następujących:

- Programy obsługi wyjścia mogą być zapisywane tylko w języku asemblacji.
- Programy obsługi wyjścia muszą być ponownie uruchamiane i zdolne do działania w dowolnym miejscu w pamięci masowej.
- Programy obsługi wyjścia muszą przywrócić środowisko w momencie wyjścia do tego wpisu i muszą zwolnić wszystkie uzyskane pamięci.
- Programy obsługi wyjścia nie mogą WAIT, ani wydawać ESTAE ani SPIEs.
- Programy obsługi wyjścia są zwykle wywoływane tak, jakby przez z/OS LINK w:
  - Nieautoryzowany stan programu problemu
  - Podstawowy tryb sterowania przestrzenią adresową
  - Tryb niepamięciowy
  - Tryb bez dostępu do rejestru
  - 31-bitowy tryb adresowania
  - Tryb TCB-PRB
- W przypadku użycia przez aplikację CICS wyjście jest wywoływane przez odsyłacz EXEC CICS LINK i musi być zgodne z konwencjami programowymi produktu CICS . Parametry są przekazywane przez wskaźniki (adresy) w obszarze komunikacji CICS (COMMAREA).

  Chociaż programy obsługi wyjścia użytkownika nie są zalecane, mogą również korzystać z wywołań funkcji API produktu CICS , z zachowaniem następującej ostrożności:

  - Nie należy wydawać punktów synchronizacji, ponieważ wyniki mogą wpływać na jednostki pracy zadeklarowane przez agenta MCA.
  - Nie należy aktualizować żadnych zasobów sterowanych przez menedżera zasobów innego niż IBM MQ for z/OS, w tym również tych, które są kontrolowane przez serwer transakcji CICS .

  Dla kanałów z parametrem CONVERT = YES wyjście jest ładowane z zestawu danych przywoływanego przez instrukcję CSQXLIB DD. Dostarczone wyjścia MQ: CSQCBDCI i CSQCBDCO dla mostu IBM MQ CICS są w SCSQAUTH.

Pisanie programu obsługi wyjścia konwersji danych dla programu IBM i

Informacje na temat kroków, które należy wziąć pod uwagę podczas pisania programów obsługi wyjścia konwersji danych MQ dla produktu IBM i.

Wykonaj następujące kroki:

1. Podaj nazwę formatu wiadomości. Nazwa musi mieścić się w polu *Format* deskryptora MQMD. Nazwa *Format* nie może zawierać początkowych odstępów, a końcowe spacje są ignorowane. Nazwa obiektu nie może zawierać więcej niż osiem znaków, ponieważ *Format* ma tylko osiem znaków długości. Pamiętaj, aby używać tej nazwy za każdym razem, gdy wysyłany jest komunikat (w naszym przykładzie używany jest format nazwy).
2. Utwórz strukturę, która będzie reprezentowana przez komunikat. Przykład można znaleźć w sekcji poprawna składnia .
3. Uruchom tę strukturę za pomocą komendy CVTMQMDTA w celu utworzenia fragmentu kodu dla wyjścia konwersji danych.

   Funkcje generowane przez komendę CVTMQMDTA używają makr, które są dostarczane w pliku QMQM/H (AMQSVMHA). Makra te są napisane przy założeniu, że wszystkie struktury są spakowane, a jeśli tak nie jest, należy je zmienić.
4. Należy wykonać kopię dostarczonego pliku źródłowego szkieletu, QMQMSAMP/QCSRC (AMQSVFC4) i zmienić jego nazwę. (W naszym przykładzie użyto nazwy EXIT_MOD.)
5. Znajdź następujące pola komentarza w pliku źródłowym i wstaw kod zgodnie z opisem:

   a. W celu zakończenia pliku źródłowego, pole komentarza rozpoczyna się od:

```
/* Insert the functions produced by the data-conversion exit */
```

W tym miejscu wstaw fragment kodu wygenerowany w kroku “3” na stronie 1084.

b. W pobliżu środka pliku źródłowego, pole komentarza rozpoczyna się od:

```
/* Insert calls to the code fragments to convert the format's */
```

Następuje to za pomocą skomentowanego wywołania funkcji `ConverttagSTRUCT`.

Zmień nazwę funkcji na nazwę funkcji, która została dodana w kroku “5.a” na stronie 1084. Usuń znaki komentarza, aby aktywować funkcję. Jeśli istnieje kilka funkcji, utwórz połączenia dla każdego z nich.

c. W pobliżu 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 1084.

Jeśli komunikat zawiera dane znakowe, wygenerowany kod wywołuje MQXCNVC; można to rozwiązać, powiążając 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/dowiązać program.

W przypadku aplikacji innych niż wielowątkowe należy użyć następujących elementów:

```
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 ładnym obiekcie musi następować _R. Użyj biblioteki LIBMQM_R, aby rozstrzygnąć wywołania do MQXCNVC. Zarówno obiekty możliwe do ładowania 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 przypadku produkcji programy obsługi wyjścia konwersji danych były przechowywane w bibliotece QSYS.

Uwaga:

1. Jeśli funkcja CVTMQMDTA używa spakowanych struktur, wszystkie aplikacje produktu IBM MQ muszą używać kwalifikatora _Packed.
2. Programy obsługi wyjścia konwersji danych muszą być ponownie wejściowane.
3. MQXCNVC jest jedynym wywołaniem MQI, które może zostać wysłane z wyjścia konwersji danych.
4. Skompiluj program obsługi wyjścia z opcją kompilatora profilu użytkownika ustawioną na *USER, tak aby program obsługi wyjścia był uruchamiany z uprawnieniami użytkownika.

5. Włączenie pamięci teraprzestrzeni jest wymagane dla wszystkich wyjść użytkownika z programem IBM MQ for IBM i ; podaj TERASPACE (*YES *TSIFC) w komendach CRTCMOD i CRTBNDC.

Pisanie programu obsługi wyjścia konwersji danych dla programu IBM MQ for z/OS

Informacje na temat kroków, które należy wziąć pod uwagę podczas zapisywania programów obsługi wyjścia konwersji danych dla produktu IBM MQ for z/OS.

Wykonaj następujące kroki:

1. Zabierz dostarczony szkielet źródła CSQ4BAX9 (w środowiskach innych niżCICS) lub CSQ4CAX9 (dla CICS) jako punkt początkowy.
2. Uruchom program narzędziowy CSQUCVX.
3. Postępuj zgodnie z instrukcjami w prologu CSQ4BAX9 lub CSQ4CAX9 , aby włączyć procedury wygenerowane przez program narzędziowy CSQUCVX, w takiej kolejności, w jakiej struktury występują w komunikacie, który ma zostać przekształty.
4. Program narzędziowy zakłada, że struktury danych nie są spakowane, że dorozumiane wyrównanie danych jest uhonorowane oraz że struktury rozpoczynają się na granicy pełnej słowa, a bajty są pomijane w zależności od potrzeb (między identyfikatorem a wersją w przykładzie w polu Ważna składnia). Jeśli struktury są spakowane, pomiń wygenerowane makra CMQXCALA. Dlatego należy rozważyć zadeklarowanie struktur w taki sposób, aby wszystkie pola zostały nazwane i nie zostały pominięte żadne bajty. W tym przykładzie w polu poprawnej składninależy 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. Chociaż wyjście konwertuje tyle pełnych pól, jak to tylko możliwe, błąd powoduje, że do aplikacji zwracany jest nieprzekształcony komunikat. Jeśli chcesz, aby krótkie bufory wejściowe były przekształcane w miarę możliwości, w tym pola częściowe, zmień wartość TRUNC= na makro CSQXCDFA na YES: nie jest zwracany błąd, więc aplikacja otrzymuje przekształcony komunikat. Aplikacja musi obsługiwać obcinanie.
6. Dodaj wymagany specjalny kod przetwarzania specjalnego.
7. Zmień nazwę programu na nazwę formatu danych.
8. Skompiluj i dowiązaj-edytuj swój program jak program wsadowy (chyba, że jest on przeznaczony do użytku z aplikacjami CICS). Makra znajdujące się w kodzie wygenerowanym przez program narzędziowy znajdują się w bibliotece **thlqual**.SCSQMACS.

   Jeśli komunikat zawiera dane znakowe, wygenerowany kod wywołuje komendę MQXCNVC. Jeśli program obsługi wyjścia korzysta z tego wywołania, należy go edytować za pomocą programu zewnętrznego CSQASTUB wyjścia. Kod pośredniczący jest niezależny od języka i jest niezależny od środowiska. Alternatywnie można załadować kod pośredniczący dynamicznie przy użyciu nazwy wywołania dynamicznego CSQXCNVC. Więcej informacji zawiera sekcja “Dynamiczne wywoływanie kodu pośredniczącego produktu IBM MQ” na stronie 1136.

   Umieść moduł edytowany łączem w bibliotece ładowania aplikacji i w zestawie danych, do którego odwołuje się instrukcja CSQXLIB DD procedury zadania uruchomionej przez inicjatora kanału.
9. Jeśli wyjście jest używane przez aplikacje produktu CICS , kompiluj i edytuj je, tak jak program użytkowy CICS , w tym CSQASTUB, jeśli jest to wymagane. Umieść go w bibliotece programu aplikacji CICS . Zdefiniuj program na 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 1086), nie są one potrzebne do edycji łącza ani do działania samego wyjścia konwersji danych (patrz kroki “8” na stronie 1086 i “9” na stronie 1086).

Informacje na temat konwersji danych w ramach mostu IBM MQ - IMS znajdują się w sekcji “Pisanie aplikacji mostu IMS” na stronie 75 .

Linux UNIX *Zapisywanie wyjścia konwersji danych dla systemu IBM MQ w systemach UNIX and Linux*

Informacje na temat kroków, które należy wziąć pod uwagę podczas zapisywania programów obsługi wyjścia konwersji danych dla systemu IBM MQ w systemach UNIX and Linux .

Wykonaj następujące kroki:

1. Podaj nazwę formatu wiadomości. Nazwa musi mieścić się w polu *Format* deskryptora MQMD i musi być podana wielkimi literami, na przykład MYFORMAT. Nazwa *Format* nie może zawierać odstępów początkowych. Odstępy końcowe są ignorowane. Nazwa obiektu nie może zawierać więcej niż osiem znaków, ponieważ *Format* ma tylko osiem znaków długości. Pamiętaj, aby używać tej nazwy za każdym razem, gdy wysyłany jest komunikat.

   Jeśli wyjście konwersji danych jest używane w środowisku wielowątkowym, obiekt ładujący musi być poprzedzony przez _r, aby wskazać, że jest to wersja wielowątkowa.
2. Utwórz strukturę, która będzie reprezentowana przez komunikat. Przykład można znaleźć w sekcji poprawna składnia .
3. Tę strukturę należy uruchomić za pomocą komendy `crtmqcvx` , aby utworzyć fragment kodu dla wyjścia konwersji danych.

   Funkcje generowane przez komendę `crtmqcvx` używają makr, które zakładają, że wszystkie struktury są spakowane; należy je zmienić, jeśli nie jest to przypadek.
4. Skopiuj dostarczony plik źródłowy szkieletu, zmieniając jego nazwę na nazwę formatu komunikatu ustawionego w kroku “1” na stronie 1087. Plik źródłowy szkieletu, a także kopia, są tylko do odczytu.

   Plik źródłowy szkieletu 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* oznacza katalog najwyższego poziomu, w którym jest zainstalowany produkt IBM MQ . Upewnij się, że ścieżka zawiera punkty ścieżki do tego katalogu, aby pobrać ten plik.

   Plik amqsvmha.h zawiera makra, które są używane przez kod wygenerowany przez komendę `crtmqcvx` . Jeśli struktura, która ma zostać przekształcona, zawiera dane znakowe, te makra wywołują MQXCNVC.
7. Znajdź następujące pola komentarza w pliku źródłowym i wstaw kod zgodnie z opisem:
   a. W celu zakończenia pliku źródłowego, pole komentarza rozpoczyna się od:

      ```
      /* Insert the functions produced by the data-conversion exit */
      ```

      W tym miejscu wstaw fragment kodu wygenerowany w kroku “3” na stronie 1087.
   b. W pobliżu środka pliku źródłowego, pole komentarza rozpoczyna się od:

      ```
      /* Insert calls to the code fragments to convert the format's */
      ```

      Następuje to za pomocą skomentowanego wywołania funkcji `ConverttagSTRUCT`.

      Zmień nazwę funkcji na nazwę funkcji, która została dodana w kroku “7.a” na stronie 1087. Usuń znaki komentarza, aby aktywować funkcję. Jeśli istnieje kilka funkcji, utwórz połączenia dla każdego z nich.
   c. W pobliżu 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 1087.

8. Skompiluj wyjście jako bibliotekę współużytkowaną, używając komendy MQStart jako punktu wejścia. Aby to zrobić, należy zapoznać się z “Kompilowanie wyjść konwersji danych w systemach UNIX and Linux” na stronie 1088.
9. Umieść dane wyjściowe w katalogu wyjściowym. Domyślnym katalogiem wyjścia jest `/var/mqm/exits` dla 32-bitowych systemów i `/var/mqm/exits64`dla systemów 64-bitowych. Te katalogi można zmienić w pliku qm.ini lub mqclient.ini . Ta ścieżka może być ustawiona dla każdego menedżera kolejek, a wyjście jest wyszukiwawcze tylko w tej ścieżce lub ścieżkach.

Uwaga:

1. Jeśli produkt `crtmqcvx` używa spakowanych struktur, wszystkie aplikacje produktu IBM MQ muszą być kompilowane w ten sposób.
2. Programy obsługi wyjścia konwersji danych muszą być ponownie wejściowane.
3. MQXCNVC jest jedynym wywołaniem MQI, które może zostać wysłane z wyjścia konwersji danych.

Linux UNIX *Kompilowanie wyjść konwersji danych w systemach UNIX and Linux*

Przykłady kompilacji wyjścia konwersji danych w systemach UNIX and Linux .

Na wszystkich platformach punktem wejścia do modułu jest MQStart.

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

AIX

AIX

Skompiluj kod źródłowy wyjścia, wydając jedną z następujących komend:

Aplikacje 32-bitowe

Niewątkowa

```
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

Niewątkowa

```
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:

31-bitowe aplikacje

Niewątkowa

```
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

Niewątkowa

```
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

Niewątkowa

```
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
```

Solaris

Solaris

Skompiluj kod źródłowy wyjścia, wydając jedną z następujących komend:

Aplikacje 32-bitowe

platforma SPARC

```
cc -xarch=v8plus -KPIC -mt -G -o /var/mqm/exits/MYFORMAT \
   MYFORMAT.c -I MQ_INSTALLATION_PATH/inc -R/usr/lib/32 -lsocket -lnsl -ldl
```

platformax86-64

```
cc -xarch=386 -KPIC -mt -G -o /var/mqm/exits/MYFORMAT \
   MYFORMAT.c -I MQ_INSTALLATION_PATH/inc -R/usr/lib/32 -lsocket -lnsl -ldl
```

Aplikacje 64-bitowe

platforma SPARC

```
cc -xarch=v9 -KPIC -mt -G -o /var/mqm/exits64/MYFORMAT \
   MYFORMAT.c -I MQ_INSTALLATION_PATH/inc -R/usr/lib/64 -lsocket -lnsl -ldl
```

platformax86-64

```
cc -xarch=amd64 -KPIC -mt -G -o /var/mqm/exits64/MYFORMAT \
   MYFORMAT.c -I MQ_INSTALLATION_PATH/inc -R/usr/lib/64 -lsocket -lnsl -ldl
```

Windows *Zapisywanie wyjścia konwersji danych dla IBM MQ for Windows*

Informacje na temat kroków, które należy wziąć pod uwagę podczas zapisywania programów obsługi wyjścia konwersji danych dla produktu IBM MQ for Windows.

Wykonaj następujące kroki:

1. Podaj nazwę formatu wiadomości. 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 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 . Należy wykonać kopię tego pliku i zmienić jej nazwę na przykład na MYFORMAT.DEF. Upewnij się, że nazwa biblioteki DLL jest tworzona i ma nazwę określoną w polu MYFORMAT.DEF jest taka sama. Zastąp nazwę FORMAT1 w polu MYFORMAT.DEF z nową nazwą formatu.

   Pamiętaj, aby używać tej nazwy za każdym razem, gdy wysyłany jest komunikat.

2. Utwórz strukturę, która będzie reprezentowana przez komunikat. Przykład można znaleźć w sekcji poprawna składnia .

3. Tę strukturę należy uruchomić za pomocą komendy `crtmqcvx` , aby utworzyć fragment kodu dla wyjścia konwersji danych.

   Funkcje generowane przez komendę CRTMQCVX używają makr, które są napisane przy założeniu, że wszystkie struktury są spakowane; należy je zmienić, jeśli nie jest to przypadek.

4. Skopiuj dostarczony plik źródłowy szkieletu amqsvfc0.c, zmieniając nazwę na nazwę formatu komunikatu ustawionego w kroku “1” na stronie 1090.

   Plik amqsvfc0.c znajduje się w katalogu *MQ_INSTALLATION_PATH*\Tools\C\Samples , gdzie *MQ_INSTALLATION_PATH* jest katalogiem, w którym zainstalowano produkt IBM MQ . Domyślny katalog instalacyjny to C:\Program Files\IBM\MQ.

   Szkielet zawiera przykładowy plik nagłówkowy amqsvmha.h w katalogu *MQ_INSTALLATION_PATH*\Tools\C\include . Upewnij się, że ścieżka zawiera punkty ścieżki do tego katalogu, aby pobrać ten plik.

   Plik amqsvmha.h zawiera makra, które są używane przez kod wygenerowany przez komendę CRTMQCVX. Jeśli struktura, która ma zostać przekształcona, zawiera dane znakowe, te makra wywołują MQXCNVC.

5. Znajdź następujące pola komentarza w pliku źródłowym i wstaw kod zgodnie z opisem:

   a. W celu zakończenia pliku źródłowego, pole komentarza rozpoczyna się od:

   ```
   /* Insert the functions produced by the data-conversion exit */
   ```

   W tym miejscu wstaw fragment kodu wygenerowany w kroku “3” na stronie 1090.

   b. W pobliżu środka pliku źródłowego, pole komentarza rozpoczyna się od:

```
/* Insert calls to the code fragments to convert the format's */
```

Następuje to za pomocą skomentowanego wywołania funkcji ConverttagSTRUCT.

Zmień nazwę funkcji na nazwę funkcji, która została dodana w kroku “5.a” na stronie 1090. Usuń znaki komentarza, aby aktywować funkcję. Jeśli istnieje kilka funkcji, utwórz połączenia dla każdego z nich.

c. W pobliżu 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 1090.

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 program obsługi wyjścia jako plik DLL.
8. Umieść dane wyjściowe w podkatalogu wyjścia poniżej katalogu danych programu IBM MQ . Domyślnym katalogiem na potrzeby instalowania wyjść w systemach 32-bitowych jest *MQ_DATA_PATH*\Exits , a dla systemów 64-bitowych jest to *MQ_DATA_PATH*\Exits64

Ścieżka używana do wyszukiwania wyjść konwersji danych jest podana w rejestrze. Folder rejestru jest następujący:

```
HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WebSphere
MQ\Installation\MQ_INSTALLATION_NAME\Configuration\ClientExitPath\
```

a kluczem rejestru jest: ExitsDefaultPath. Ta ścieżka może być ustawiona dla każdego menedżera kolejek, a wyjście jest wyszukiwawcze tylko w tej ścieżce lub ścieżkach.

Uwaga:

1. Jeśli komenda CRTMQCVX używa spakowanych struktur, wszystkie aplikacje produktu IBM MQ muszą być kompilowane w ten sposób.
2. Programy obsługi wyjścia konwersji danych muszą być ponownie wejściowane.
3. MQXCNVC jest jedynym wywołaniem MQI, które może zostać wysłane z wyjścia konwersji danych.

Windows *Wyjdź i przełączaj pliki ładowania w systemach operacyjnych Windows*

Procesy menedżera kolejek produktu IBM WebSphere MQ for Windows 7.5 są 32-bitowe. W rezultacie w przypadku używania aplikacji 64-bitowych niektóre typy plików ładowania wyjścia i ładowania z przełącznikami XA również muszą mieć wersję 32-bitową dostępną do użycia przez menedżer kolejek. Jeśli wymagana jest 32-bitowa wersja wyjścia lub plik ładowania przełącznika XA, a nie jest on dostępny, to wywołanie lub wykonanie odpowiedniej komendy API nie powiedzie się.

W polu qm.ini file dla parametru *ExitPath*obsługiwane są dwa atrybuty. Są to ExitsDefaultPath= *MQ_INSTALLATION_PATH*\exits i ExitsDefaultPath64= *MQ_INSTALLATION_PATH*\exits64. *MQ_INSTALLATION_PATH* reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ . Dzięki temu możliwe będzie odnalezcie odpowiedniej biblioteki. Jeśli wyjście jest używane w klastrze IBM MQ , to zapewnia również, że można znaleźć odpowiednią bibliotekę w systemie zdalnym.

Poniższa tabela zawiera listę różnych typów plików ładowania wyjścia i przełączników oraz informacje o tym, czy są wymagane 32-bitowe, czy 64-bitowe wersje, czy są używane aplikacje 32-lub 64-bitowe:

Typy plików	Aplikacje 32-bitowe	Aplikacje 64-bitowe
wyjście funkcji API	32-i 64-bitowy	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 usługi instalowalnej	64-bitowe	64-bitowe
Wyjście WLM klastra	64-bitowe	64-bitowe
Wyjście routingu publikowania/ subskrypcji	64-bitowe	64-bitowe
Pliki ładowania przełączników bazy danych	32-i 64-bitowy	64-bitowe
Zewnętrzne biblioteki AX menedżera transakcji zewnętrznych	32 bity	64-bitowe
Wyjście przed nawiązaniem połączenia	32 bity	64-bitowe

Odwołanie do definicji połączenia przy użyciu wyjścia wstępnego z połączeniem z repozytorium

Produkt IBM MQ MQI clients można skonfigurować w taki sposób, aby wyszukać repozytorium w celu uzyskania definicji połączeń przy użyciu biblioteki wyjścia wstępnego połączenia.

Wprowadzenie

Aplikacja kliencka może łączyć się z menedżerem kolejek przy użyciu tabel definicji kanału klienta (CCDT). Zwykle plik CCDT znajduje się na centralnym serwerze plików sieciowych i ma do niego odniesienie klientów. Ponieważ zarządzanie różnymi aplikacjami klienckim odwołujących się do pliku CCDT jest trudne, elastycznym podejściem jest przechowywanie definicji klientów w repozytorium globalnym, takich jak katalog LDAP, rejestr i repozytorium produktu WebSphere lub dowolne inne repozytorium. Przechowywanie definicji połączeń klienta w repozytorium ułatwia zarządzanie definicjami połączenia klienckiego, a aplikacje mogą uzyskiwać dostęp do poprawnych i najbardziej aktualnych definicji połączeń klientów.

Podczas wykonywania wywołania MQCONN/X program IBM MQ MQI client ładuje określoną bibliotekę wyjścia do połączenia z określoną aplikacją, a następnie wywołuje funkcję wyjścia w celu pobrania definicji połączeń. Pobrane definicje połączeń są następnie używane do nawiązania połączenia z menedżerem kolejek. Szczegóły dotyczące biblioteki obsługi wyjścia i funkcji do wywołania są określone w pliku konfiguracyjnym mqclient.ini .

Składnia

`void MQ_PRECONNECT_EXIT` (parametry*pExitParms*, *pQMgrName*, *ppConnectOpts*, *pCompCode*, *pReason*);

Parametry

Parametry pExit
Typ: PMQNXP wejście/wyjście

Struktura parametru wyjścia **PreConnection** .

Struktura jest przydzielana i obsługiwana przez program wywołujący wyjście.

Nazwa pQMgr
Typ: input/output PMQCHAR

Nazwa menedżera kolejek.

Na wejściu parametr ten jest łańcuchem filtru dostarczonym do wywołania MQCONN API za pomocą parametru **QMgrName** . To pole może być puste, jawne lub zawierać określone znaki wieloznaczne. Pole zostanie zmienione przez wyjście. Parametr ma wartość NULL, gdy wyjście jest wywoływane z MQXR_TERM.

ppConnectOpts
Wpisz: ppConnectOpts input/output

Opcje, które sterują działaniem MQCONNX.

Jest to wskaźnik do struktury opcji połączenia MQCNO, która steruje działaniem wywołania interfejsu API MQCONN. Parametr ma wartość NULL, gdy wyjście jest wywoływane z MQXR_TERM. Klient MQI zawsze udostępnia strukturę MQCNO do wyjścia, nawet jeśli nie została ona pierwotnie udostępniona przez aplikację. Jeśli aplikacja udostępnia strukturę MQCNO, klient tworzy duplikat w celu przekazania go do wyjścia, w którym jest on modyfikowany. Klient zachowuje prawo własności do obiektu MQCNO.

Tabela MQCD, do której odwołuje się parametr MQCNO, ma pierwszeństwo przed każdą definicją połączenia udostępnionej przez tablicę. Klient używa struktury MQCNO do łączenia się z menedżerem kolejek, a pozostałe są ignorowane.

Kod pComp
Typ: PMQLONG input/output

Kod zakończenia.

Wskaźnik do obiektu MQLONG, który odbiera kod zakończenia wyjścia. Musi to być jedna z następujących wartości:

- MQCC_OK -pomyślne zakończenie
- MQCC_WARNING -ostrzeżenie (częściowe zakończenie)
- MQCC_FAILED -wywołanie nie powiodło się

pReason
Typ: PMQLONG input/output

Przyczyna kwalifikowania kodu pComp.

Wskaźnik do obiektu MQLONG, który odbiera kod przyczyny wyjścia. Jeśli kodem zakończenia jest MQCC_OK, jedyną poprawną wartością jest:

- MQRC_NONE-(0, x '000') Nie ma powodu do zgłaszania.

Jeśli kod zakończenia to MQCC_FAILED lub MQCC_WARNING, to funkcja wyjścia może ustawić pole kodu przyczyny na 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*/
```

Pisanie i kompilowanie wyjść publikowania

Istnieje możliwość skonfigurowania wyjścia publikowania w menedżerze kolejek w celu zmiany treści opublikowanego komunikatu, zanim zostanie on odebrany przez subskrybentów. Można również zmienić nagłówek komunikatu lub nie dostarczyć go do subskrypcji.

Uwaga: Wyjścia publikowania nie są obsługiwane w produkcie z/OS.

Wyjścia publikowania można używać do sprawdzania i modyfikowania komunikatów dostarczanych do subskrybentów:

- Sprawdzanie treści komunikatu publikowanego dla każdego subskrybenta
- Modyfikowanie treści komunikatu publikowanego dla każdego subskrybenta
- Zmień kolejkę, do której jest wstawiany komunikat
- Zatrzymaj dostarczanie komunikatu do subskrybenta

Zapisywanie wyjścia publikowania

Wykonaj kroki opisane w sekcji “Zapisywanie wyjść i instalowalnych usług w systemach UNIX, Linux i Windows” na stronie 1033, aby ułatwić pisanie i kompilowanie wyjścia.

Dostawca wyjścia publikowania definiuje to, co robi wyjście. Wyjście musi jednak być zgodne z regułami zdefiniowanymi w MQPSXP.

Produkt IBM MQ nie udostępnia implementacji punktu wejścia MQ_PUBLISH_EXIT. Udostępnia deklarację typu C języka C. Użyj typedef, aby poprawnie zadeklarować parametry do wyjścia napisanego przez użytkownika. 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 ramach procesu menedżera kolejek w wyniku następujących operacji:

- Operacja publikowania, w której komunikat jest dostarczany do jednego lub większej liczby subskrybentów
- Operacja Subskrybuj, w której jest dostarczany jeden lub więcej zachowanych komunikatów.
- Operacja żądania subskrypcji, w której jeden lub więcej zatrzymanych komunikatów jest dostarczanych

Jeśli wyjście publikowania jest wywoływane dla połączenia, po raz pierwszy jest wywoływany kod *ExitReason* produktu MQXR_INIT . Przed rozłączonym połączeniem po użyciu wyjścia publikowania, wyjście jest wywoływane z kodem *ExitReason* produktu MQXR_TERM.

Jeśli wyjście publikowania jest skonfigurowane, ale nie można go załadować po uruchomieniu menedżera kolejek, operacje komunikatów publikowania/subskrypcji są blokowane dla menedżera kolejek. Należy naprawić problem lub zrestartować menedżer kolejek, aby przesyłanie komunikatów w trybie publikowania/subskrypcji było ponownie włączone.

Każde połączenie IBM MQ , które wymaga wyjścia publikowania, może nie zostać załadowane lub zainicjowane wyjście. Jeśli operacja wyjścia nie powiedzie się lub zainicjowano, operacje publikowania/subskrypcji wymagające wyjścia publikowania są wyłączone dla tego połączenia. Operacje nie powiodą się z kodem przyczyny IBM MQ MQRC_PUBLISH_EXIT_ERROR.

Kontekst, w którym wywoływane jest wyjście publikowania, jest połączeniem przez aplikację z menedżerem kolejek. Obszar danych użytkownika jest obsługiwany przez menedżera kolejek dla każdego połączenia, które wykonuje operacje publikowania. Wyjście może zachować informacje w obszarze danych użytkownika dla każdego połączenia.

Wyjście publikowania może korzystać z niektórych wywołań MQI. Mogą one używać tylko tych wywołań MQI, które manipulują właściwościami komunikatów. Połączenia są następujące:

- MQBUFMH
- MQCRTMH
- MQDLTMH
- MQDLTMP
- MQMHBUF
- MQINQMP
- MQSETMP

Jeśli wyjście publikowania zmieni docelowy menedżer kolejek lub nazwę kolejki, nie jest przeprowadzane żadne nowe sprawdzanie uprawnień.

Kompilowanie wyjścia publikowania

Wyjście publikowania jest biblioteką ładowaną dynamicznie; może być ona pomyślana jako wyjście kanału. Więcej informacji na temat kompilowania wyjść zawiera sekcja "Zapisywanie wyjść i instalowalnych usług w systemach UNIX, Linux i Windows" na stronie 1033.

Przykładowe wyjście publikowania

Przykładowy program obsługi wyjścia nosi nazwę amqspse0.c. Zapisuje on inny komunikat w pliku dziennika w zależności od tego, czy została wywołana dla operacji inicjowania, publikowania lub zakończenia. Demonstruje również użycie pola obszaru użytkownika obsługi wyjścia w celu odpowiedniego przydzielania i swobodnej pamięci masowej.

Konfigurowanie wyjść publikowania

Aby skonfigurować wyjście publikowania, należy zdefiniować niektóre atrybuty.

W systemach Windows i Linux można użyć eksploratora IBM MQ w celu zdefiniowania atrybutów. Atrybuty są definiowane na stronie właściwości menedżera kolejek w obszarze Publikowanie/subskrypcja.

Aby skonfigurować wyjście publikowania w pliku qm.ini w systemach UNIX and Linux , należy utworzyć sekcję o nazwie PublishSubscribe. Sekcja PublishSubscribe zawiera następujące atrybuty:

PublishExitŚcieżka = [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.

PublishExitFunkcja = *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.

PublishExitDane = *łańcuch*
Jeśli menedżer kolejek wywołuje wyjście publikowania, przekazuje on strukturę MQPSXP jako dane wejściowe. Dane określone za pomocą atrybutu **PublishExitData** są dostępne w polu *ExitData* struktury. Łańcuch może mieć długość do MQ_EXIT_DATA_LENGTH znaków. Wartością domyślną są 32 puste znaki.

Pisanie i kompilowanie wyjść obciążenia klastra

Napisz program obsługi wyjścia obciążenia klastra, aby dostosować zarządzanie obciążeniem klastrów. Podczas kierowania komunikatów można wziąć pod uwagę koszt korzystania z kanału o różnych porach dnia lub treści komunikatu. Są to czynniki, które nie są brane pod uwagę przy użyciu standardowego algorytmu zarządzania obciążeniem.

W większości przypadków algorytm zarządzania obciążeniem jest wystarczający dla potrzeb użytkownika. Aby jednak można było udostępnić własny program obsługi wyjścia użytkownika w celu dostosowania zarządzania obciążeniem, program IBM MQ będzie zawierał wyjście użytkownika, wyjście obciążenia klastra.

Użytkownik może mieć pewne konkretne informacje na temat sieci lub komunikatów, których można użyć w celu wpływania na równoważenie obciążenia. Może się okazać, że są to kanały o dużej pojemności lub tanie trasy sieciowe, lub też można kierować komunikaty w zależności od ich zawartości. Użytkownik może zdecydować się na napisanie programu obsługi wyjścia obciążenia klastra lub użyć jednej z nich dostarczonych przez osobę trzecią.

Wyjście obciążenia klastra jest wywoływane podczas uzyskiwania dostępu do kolejki klastra. Jest on wywoływany przez produkty MQOPEN, MQPUT1 i MQPUT.

Docelowy menedżer kolejek wybrany w czasie MQOPEN jest stały, jeśli określono 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 programu MQPUT . Jeśli docelowy menedżer kolejek jest niedostępny lub wystąpił błąd w czasie, gdy komunikat nadal znajduje się w kolejce transmisji, to wyjście jest ponownie wywoływane. Zostanie wybrany nowy docelowy menedżer kolejek. Jeśli kanał komunikatów nie powiedzie się, gdy komunikat jest przesyłany, a 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, to wyjście obciążenia klastra nie jest wywoływane.

Różne dane są przekazywane do wyjścia obciążenia klastra w strukturze parametru wyjścia, MQWXP:

- Struktura definicji komunikatu MQMD.
- Parametr długości komunikatu.
- Kopia komunikatu lub część komunikatu.

Na platformach innych niżz/OS , jeśli używany jest produkt CLWLMode=FAST, każdy proces systemu operacyjnego ładuje własną kopię wyjścia. Różne połączenia z menedżerem kolejek mogą spowodować wywołanie różnych kopii wyjścia. Jeśli wyjście zostanie uruchomione w domyślnym trybie bezpiecznym, CLWLMode=SAFE, pojedyncza kopia wyjścia działa w osobnym procesie.

Zapisywanie wyjść obciążenia klastra

z/OS Informacje na temat zapisywania wyjść obciążenia klastra dla produktu z/OSmożna znaleźć w sekcji “Programowanie wyjścia obciążenia klastra dla IBM MQ for z/OS” na stronie 1098.

Multi W przypadku wielu platform wyjścia obciążenia klastra nie mogą używać wywołań MQI. W innych aspektach reguły pisania i kompilowania programów obsługi wyjścia obciążenia klastra są podobne do reguł, które mają zastosowanie do programów obsługi wyjścia kanału. Wykonaj kroki opisane w sekcji “Zapisywanie wyjść i instalowalnych usług w systemach UNIX, Linux i Windows” na stronie 1033, a następnie użyj przykładowego programu “Przykładowe wyjście obciążenia klastra” na stronie 1097 , aby pomóc w napisanie i skompilowaniu wyjścia.

Więcej informacji na temat wyjść kanału zawiera sekcja “Pisanie programów obsługi wyjścia kanału” na stronie 1062.

Konfigurowanie wyjść obciążenia klastra

Należy podać wyjścia obciążenia klastra 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

Produkt 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 może zostać znaleziona w bibliotece `thlqual.SCSQASMS`. Wersja produktu C nosi nazwę CSQ4BCF1 i może zostać znaleziona w bibliotece `thlqual.SCSQC37S`. `thlqual` to kwalifikator wysokiego poziomu biblioteki docelowej dla zestawów danych IBM MQ w instalacji.

Multi **IBM MQ for Multiplatforms**

Przykładowy program obsługi wyjścia obciążenia klastra jest dostarczany w języku C i jest nazywany `amqswlm0.c`. Można go znaleźć w:

Tabela 148. Przykładowe położenie programu obsługi wyjścia obciążenia klastra dla wielu platform

Platforma	Ścieżka do pliku
AIX AIX Solaris Solaris	*MQ_INSTALLATION_PATH*/samp
Windows Windows	*MQ_INSTALLATION_PATH* \Tools\c\Samples
IBM i IBM i	Biblioteka qmqm

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Ten przykładowy program obsługi wyjścia kieruje wszystkie komunikaty do konkretnego menedżera kolejek, chyba że ten menedżer kolejek stanie się niedostępny. Reaguje ona na awarię menedżera kolejek przez kierowanie komunikatów do innego menedżera kolejek.

Wskaż, do którego menedżera kolejek, do którego mają być wysyłane komunikaty. Podaj nazwę kanału odbierającego klastry w atrybucie CLWLDATA w definicji menedżera kolejek. Na przykład:

```
ALTER QMGR CLWLDATA(' my-cluster-name. my-queue-manager ')
```

Aby włączyć wyjście, podaj jego pełną ścieżkę i nazwę w atrybucie CLWLEXIT :

Linux UNIX W systemie UNIX 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 języka programowania używanej wersji.

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 są puste-dopełnione w prawo, jeśli to konieczne.
- Użyj komendy CL:

  ```
  CHGMQM MQMNAME( qmgrname ) CLWLEXIT(' library /AMQSWLM')
  ```

Teraz, zamiast korzystać z dostarczonego algorytmu zarządzania obciążeniem, program IBM MQ wywołuje to wyjście, aby skierować wszystkie komunikaty do wybranego menedżera kolejek.

z/OS *Programowanie wyjścia obciążenia klastra dla IBM MQ for z/OS*

Wyjścia obciążenia klastra są wywoływane w taki sposób, jakby przez komendę z/OS **LINK** . Wyjścia podlegają szeregowi rygorystycznych reguł programowania. Unikaj używania większości komend SVC, które wymagają oczekiwania, lub używając STAE lub ESTAE w wyjściu obciążenia.

Wyjścia obciążenia klastra są wywoływane tak, jakby przez z/OS **LINK** w:

- Nieautoryzowany stan programu problemu
- Podstawowy tryb sterowania przestrzenią adresową
- Tryb niepamięciowy
- Tryb rejestru bez dostępu
- 31-bitowy tryb adresowania
- Klucz pamięci 8
- Maska klucza programu 8
- Klucz TCB 8

V 9.1.0 Umieść moduły edytowane przez połączenie w zestawie danych określonym za pomocą instrukcji `CSQXLIB DD` procedury uruchomionego zadania inicjatora kanału. Nazwy modułów ładowalnych są określone jako nazwy wyjścia obciążenia w definicji menedżera kolejek.

Podczas zapisywania wyjść obciążenia dla produktu IBM MQ for z/OSmają zastosowanie następujące reguły:

- Musisz pisać wyjścia w asemblerze lub w C. Jeśli używany jest język C, musi on być zgodny ze środowiskiem programistycznym systemów C dla wyjść systemowych, opisane w publikacji *z/OS C/C++ Programming Guide*(Podręcznik programowania C/C + C/C++), SC09-4765.
- Jeśli używane jest wywołanie `MQXCLWLN` , należy przejść do edycji odsyłaczy z produktem `CSQMFCLW`dostarczonym w produkcie *`thlqual`*`.SCSQLOAD`.
- Wyjścia są ładowane z nieautoryzowanych bibliotek zdefiniowanych w instrukcji `CSQXLIB DD` . Jeśli program `CSQXLIB` ma `DISP=SHR`, wyjścia mogą być aktualizowane, gdy menedżer kolejek jest

uruchomiony, z nową wersją używaną w następnym wątku MQCONN , który jest uruchamiany przez menedżer kolejek.

- Wyjścia muszą być ponownie uruchamiane i zdolne do działania w dowolnym miejscu wirtualnej pamięci masowej.
- Wyjścia muszą resetować środowisko w momencie powrotu do pozycji w pozycji.
- Wyjścia muszą zwolnić dowolną ilość pamięci masowej lub zapewnić, że pamięć masowa zostanie zwolniona przez kolejne wywołanie wyjścia.
- Żadne wywołania MQI nie są dozwolone.
- Wyjścia nie mogą używać żadnych usług systemowych, które mogłyby spowodować oczekiwanie, ponieważ oczekiwanie poważnie obniżyłoby wydajność menedżera kolejek. W związku z tym należy unikać SVC, komputera PC ani we/wy.
- Wyjścia nie mogą wydawać ESTAE ani SPIE, z wyjątkiem wszystkich podzadań, które się do nich dołączają.

Uwaga: Nie ma ograniczeń bezwzględnych co do tego, co można zrobić w wyjściu. Jednak większość programów SVC wiąże się z oczekiwaniem, dlatego należy unikać ich, z wyjątkiem następujących komend:

- **GETMAIN** / **FREEMAIN**
- **LOAD** / **DELETE**

Nie należy używać ESTAEs i ESPIEs, ponieważ ich obsługa błędów może zakłócać obsługę błędów wykonywanych przez produkt IBM MQ. Program IBM MQ może nie być w stanie naprawić błędu, albo program obsługi wyjścia nie otrzyma wszystkich informacji o błędzie.

Parametr systemowy EXITLIM ogranicza ilość czasu, przez jaki może być wykonywane wyjście. Wartość domyślna parametru EXITLIM wynosi 30 sekund. Jeśli zostanie wyświetlony kod powrotu MQRC_CLUSTER_EXIT_ERROR, 2266 X'8DA' , wyjście może być pętlą. Jeśli uważasz, że wyjście wymaga więcej niż 30 sekund do zakończenia, zwiększ wartość EXITLIM.

Budowanie aplikacji proceduralnej

Aplikację IBM MQ można napisać w jednym z kilku języków proceduralnych, a następnie uruchomić aplikację na kilku różnych platformach.

AIX Budowanie aplikacji proceduralnej w systemie AIX

Publikacje dotyczące produktu AIX opisują sposób budowania aplikacji wykonywalnych z programów, które są zapisywane.

W tym temacie opisano dodatkowe zadania oraz zmiany w standardowych zadaniach, które należy wykonać podczas budowania aplikacji produktu IBM MQ for AIX w celu uruchomienia w ramach produktu AIX. Obsługiwane są: C, C + + i COBOL. Informacje na temat przygotowywania programów w języku C + + znajdują się w sekcji Korzystanie z języka C++.

Zadania, które należy wykonać, aby utworzyć aplikację wykonywalną przy użyciu programu IBM MQ for AIX , różnią się w zależności od języka programowania, w którym jest zapisany kod źródłowy. Oprócz kodowania wywołań MQI w kodzie źródłowym, należy dodać odpowiednie instrukcje języka, aby zawierały pliki włączanych produktu IBM MQ for AIX dla używanego języka. Zaznajomić się z zawartością tych plików. Pełny opis znajduje się w sekcji “Pliki definicji danych produktu IBM MQ” na stronie 806 .

Podczas uruchamiania wielowątkowego serwera lub wielowątkowych aplikacji klienckich należy ustawić zmienną środowiskową AIXTHREAD_SCOPE = S.

AIX *Przygotowywanie programów w języku C w programie AIX*

Ten temat zawiera informacje na temat łączenia bibliotek niezbędnych do przygotowania programów w języku C w systemie AIX.

Wstępnie skompilowane 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 można znaleźć w sekcji Standardy kodowania na platformach 64-bitowych.

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest 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.

Jeśli używany jest kompilator VisualAge C/C++ dla programów w języku C++, należy uwzględnić opcję `-q namemangling=v5` , aby wszystkie symbole IBM MQ zostały rozstrzygnięte podczas tworzenia połączeń z bibliotekami.

Jeśli programy mają być używane na komputerze, na którym jest zainstalowany tylko IBM MQ MQI client for AIX , należy ponownie skompilować programy w celu połączenia ich z biblioteką klienta (`-lmqic`).

Łączenie bibliotek

Potrzebne są następujące biblioteki:

- Powiąż programy z odpowiednią biblioteką udostępnionej przez produkt IBM MQ.

  W środowisku innym niż wielowątkowe należy utworzyć odsyłacz do jednej z następujących bibliotek:

Zbiór biblioteki	Typ programu/wyjścia
libmqm.a	Serwer dla C
libmqic.a & libmqm.a	Klient dla C

  W środowisku wielowątkowym należy połączyć się z jedną z następujących bibliotek:

Zbiór biblioteki	Typ programu/wyjścia
libmqm_r.a	Serwer dla C
libmqic_r.a & libmqm_r.a	Klient dla C

  Na przykład, aby zbudować prostą wielowątkową aplikację IBM MQ z jednej jednostki kompilacji, uruchom 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.

Jeśli programy mają być używane na komputerze, na którym jest zainstalowany tylko IBM MQ MQI client for AIX , należy ponownie skompilować programy w celu połączenia ich z biblioteką klienta (`-lmqic`).

Uwaga:

1. Nie można połączyć się z więcej niż jedną biblioteką. Oznacza to, że nie można jednocześnie połączyć się z biblioteką wielowątkową i niewielowątkową.
2. Jeśli użytkownik zapisuje usługę instalowalną (więcej informacji na ten temat znajduje się w publikacji Administrowanie), należy połączyć się z biblioteką `libmqmzf.a` w aplikacji niewielowątkowej oraz z biblioteką `libmqmzf_r.a` w aplikacji wielowątkowej.
3. W przypadku tworzenia aplikacji na potrzeby zewnętrznej koordynacji za pomocą menedżera transakcji zgodnego z interfejsem XA, takiego 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) oraz biblioteki produktu `libmqz.a` w aplikacji niewielowątkowej oraz `libmqmxa_r.a` (lub `libmqmxa64_r.a`). i bibliotek produktu `libmqz_r.a` w aplikacji wielowątkowej.
4. Należy połączyć zaufane aplikacje z wątkami bibliotek produktu IBM MQ . Jednak tylko jeden wątek w zaufanej aplikacji na serwerze IBM MQ w systemach UNIX and Linux może być połączony w danym momencie.
5. Biblioteki produktu IBM MQ muszą być dowiązane przed innymi bibliotekami produktu.

AIX *Przygotowywanie programów w języku COBOL w produkcie AIX*

Te informacje są używane podczas przygotowywania programów w języku COBOL w programie AIX przy użyciu zestawów IBM COBOL Set i Micro Focus COBOL.

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

- 32-bitowe podręczniki do kopiowania 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 podręczniki do kopiowania w języku COBOL są instalowane w następującym katalogu:

  ```
  MQ_INSTALLATION_PATH/inc/cobcpy64
  ```

W poniższych przykładach ustaw zmienną środowiskową **COBCPY** na:

```
MQ_INSTALLATION_PATH/inc/cobcpy32
```

dla aplikacji 32-bitowych, oraz:

```
MQ_INSTALLATION_PATH/inc/cobcpy64
```

dla aplikacji 64-bitowych.

Musisz połączyć swój program z jednym z następujących plików bibliotecznych:

Zbiór biblioteki	Typ programu/wyjścia
libmqmcb.a	Serwer dla języka COBOL (aplikacja wielowątkowa)
libmqmcb_r.a	Serwer dla języka COBOL (aplikacja wielowątkowa)

Zbiór biblioteki	Typ programu/wyjścia
libmqicb.a	Klient dla języka COBOL (aplikacja wielowątkowa
libmqicb_r.a	Klient dla języka COBOL (aplikacja wielowątkowa)

Kompilator IBM COBOL Set lub kompilator Micro Focus COBOL można używać w zależności od programu:

- 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 zestawu IBM COBOL Set for AIX

Przykładowe programy w języku COBOL są dostarczane razem z programem IBM MQ. Aby skompilować taki program, wprowadź odpowiednią komendę z następującej listy:

32-bitowa aplikacja serwera, która nie jest wielowątkowa

```
$ cob2 -o amq0put0 amq0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb -qLIB \
-ICOBCPY_VALUE
```

32-bitowa aplikacja kliencka

```
$ cob2 -o amq0put0 amq0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb -qLIB \
-ICOBCPY_VALUE
```

32-bitowa aplikacja serwera wielowątkowego

```
$ cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqmcb_r -qLIB -ICOBCPY_VALUE
```

32-bitowa aplikacja kliencka

```
$ cob2_r -o amq0put0 amq0put0.cbl -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqicb_r -qLIB -ICOBCPY_VALUE
```

64-bitowa aplikacja serwera bez wątków

```
$ cob2 -o amq0put0 amq0put0.cbl -q64 -L MQ_INSTALLATION_PATH/lib - lmqmcb \
-qLIB -ICOBCPY_VALUE
```

64-bitowa aplikacja kliencka, która nie jest wielowątkowa

```
$ cob2 -o amq0put0 amq0put0.cbl -q64 -L MQ_INSTALLATION_PATH/lib - lmqicb \
-qLIB -ICOBCPY_VALUE
```

Aplikacja serwera 64-bitowego

```
$ cob2_r -o amq0put0 amq0put0.cbl -q64 -qTHREAD -L MQ_INSTALLATION_PATH/lib \
-lmqmcb_r -qLIB -ICOBCPY_VALUE
```

Aplikacja kliencka 64-bitowego

```
$ 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 Micro Focus COBOL

Ustaw zmienne środowiskowe przed kompilacją programu w następujący sposób:

```
export COBCPY=COBCPY_VALUE
export LIBPATH=MQ_INSTALLATION_PATH/lib:$LIBPATH
```

Aby skompilować 32-bitowy program w języku COBOL przy użyciu Micro Focus COBOL, wpisz:

- 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 wątków dla języka COBOL

```
$ cob32 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r
```

- Klient wielowątkowy dla 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, wpisz:

- 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 wątków dla języka COBOL

```
$ cob64 -xtvP amqminqx.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r
```

- Klient wielowątkowy dla 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 produktu Micro Focus COBOL.

AIX *Przygotowywanie programów aplikacji CICS w produkcie AIX*

Te informacje są używane podczas przygotowywania programów CICS w programie AIX.

Użyj modułów *switch XA switch* , aby połączyć produkt CICS z produktem IBM MQ. Więcej informacji na temat struktury przełącznika XA można znaleźć w sekcji Struktury przełącznika XA.

Dostępny jest przykładowy plik kodu źródłowego, który umożliwia tworzenie przełączników XA dla innych komunikatów transakcji. Podana nazwa modułu ładowania przełącznika znajduje się na liścieTabela 149 na stronie 1104.

Tabela 149. Podstawowy kod dla programów aplikacji CICS w systemie AIX: procedura inicjowania interfejsu XA		
Opis	**C (źródło)**	**C (exec)-dodaj do XAD.Stanza**
Procedura inicjowania interfejsu XA	amqzscix.c	*amqzsc* - CICS dla AIX

Należy użyć gotowej wersji pliku ładowania przełącznika IBM MQ *amqzsc*, który jest dostarczany wraz z produktem.

Zawsze dowiąż transakcje C z biblioteką IBM MQ wątkowo bezpieczną *libmqm_r.a.*, oraz transakcje w języku COBOL za pomocą biblioteki COBOL *libmqmcb_r.a.*.

Więcej informacji na temat obsługi transakcji CICS można znaleźć w Podręczniku administrowania systemem Administrowanie IBM MQ .

AIX *Obsługa TXSeries CICS*

IBM MQ on AIX supports TXSeries CICS using the XA interface. Upewnij się, że aplikacje produktu CICS są dowiązane do wielowątkowej wersji bibliotek produktu IBM MQ .

Programy CICS można uruchamiać za pomocą zestawu IBM COBOL Set for AIX lub Micro Focus COBOL. W poniższych sekcjach opisano różnice między uruchamianych programów CICS w produkcie 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 utworzyć połączenia wywołań MQI języka C i COBOL do tego samego regionu CICS . Większość wywołań MQI w drugim języku nie powiodła się z kodem przyczyny MQRC_H0BJ_ERR0R.

Przygotowywanie programów w języku COBOL produktu CICS przy użyciu zestawu IBM COBOL Set for AIX

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Aby użyć języka COBOL produktu 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. Konwertuj, skompiluj i dowiązaj program wpisując:

   ```
   cicstcl -l IBMCOB yourprog.ccp
   ```

Przygotowywanie programów CICS w języku COBOL za pomocą języka Micro Focus COBOL

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Aby użyć programu Micro Focus COBOL, należy wykonać następujące czynności:

1. Dodaj moduł biblioteki środowiska wykonawczego produktu IBM MQ COBOL do biblioteki środowiska wykonawczego za pomocą następującej komendy:

```
cicsmkcobol -L/usr/lib/dce -L MQ_INSTALLATION_PATH/lib \
            MQ_INSTALLATION_PATH/lib/libmqmcbrt.o -lmqe_r
```

   Uwaga: W przypadku produktu `cicsmkcobol`produkt IBM MQ nie umożliwia tworzenia wywołań MQI w języku programowania w języku C z poziomu aplikacji COBOL.

   Jeśli w istniejących aplikacjach istnieją takie wywołania, zalecane jest przeniesienie tych funkcji z aplikacji w języku COBOL do własnej biblioteki, na przykład `myMQ.so`. Po przeniesieniu funkcji nie należy dołączać biblioteki IBM MQ `libmqmcbrt.o` podczas budowania aplikacji COBOL dla produktu CICS.

   Ponadto, jeśli aplikacja COBOL nie wykonuje żadnego wywołania MQI COBOL, nie należy łączyć `libmqmz_r` z `cicsmkcobol`.

   Spowoduje to utworzenie pliku metody języka Micro Focus COBOL i włączenie biblioteki COBOL środowiska wykonawczego produktu CICS do wywoływania produktu IBM MQ w systemach UNIX and Linux .

   Uwaga: Uruchom program `cicsmkcobol` tylko wtedy, gdy instalowany jest jeden z następujących produktów:

   - Nowa wersja lub wydanie Micro Focus COBOL
   - Nowa wersja lub wydanie produktu CICS for AIX
   - Nowa wersja lub wydanie dowolnego obsługiwanego produktu bazodanowego (tylko dla transakcji w języku COBOL)
   - Nowa wersja lub wydanie produktu IBM MQ
2. Wyeksportuj następującą zmienną środowiskową:

```
COBCPY= MQ_INSTALLATION_PATH/inc export COBCPY
```

3. Konwertuj, skompiluj i dowiązaj program wpisując:

```
cicstcl -l COBOL -e yourprog.ccp
```

Przygotowywanie programów CICS C

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Zbuduj programy CICS C, korzystając ze standardowych narzędzi CICS :

1. Wyeksportuj ***jeden*** 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. Konwertuj, skompiluj i dowiązaj program wpisując:

```
cicstcl -l C amqscic0.ccs
```

Przykładowa transakcja CICS C

Przykładowe źródło C dla transakcji AIX IBM MQ jest udostępniane przez program 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

The IBM i publications describe how to build executable applications from the programs that you write, to run with IBM i on iSeries or System i systems.

W tym temacie opisano dodatkowe zadania oraz zmiany w standardowych zadaniach, które należy wykonać podczas budowania aplikacji proceduralnych IBM MQ for IBM i w celu uruchomienia w systemach IBM i . Obsługiwane są języki programowania COBOL, C, C + +, Java i RPG. Informacje na temat przygotowywania programów w języku C + + znajdują się w sekcji 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 zapisany jest kod źródłowy. Oprócz kodowania wywołań MQI w kodzie źródłowym, należy dodać odpowiednie instrukcje języka, aby uwzględnić pliki definicji danych produktu IBM MQ for IBM i dla używanego języka. Zaznajomić się z zawartością tych plików. Pełny opis znajduje się w sekcji “Pliki definicji danych produktu IBM MQ” na stronie 806 .

IBM i *Przygotowywanie programów w języku C w programie IBM i*

Produkt IBM MQ for IBM i obsługuje komunikaty o wielkości do 100 MB. Programy użytkowe napisane w języku ILE C, obsługujące komunikaty IBM MQ większe niż 16 MB, muszą używać opcji kompilatora *Teraspace* w celu przydzielenia wystarczającej ilości pamięci dla tych komunikatów.

Więcej informacji na temat opcji kompilatora języka C można znaleźć w publikacji *WebSphere Development Studio ILE C/C++ Programmer's Guide*.

Aby skompilować moduł C, można użyć komendy IBM i ,CRTCMOD. Podczas kompilowania należy upewnić się, że biblioteka zawierająca pliki włączanych (QMQM) znajduje się na liście bibliotek.

Następnie należy powiązać dane wyjściowe kompilatora z programem usługowym za pomocą komendy CRTPGM.

Tabela 150. Przykład CRTPGM w środowisku bez wątków

Komenda	Typ programu/wyjścia
`CRTPGM PGM( pgmname ) MODULE( pgmname )` `BNDSRVPGM(QMQM/LIBMQM)`	Serwer lub klient dla języka C

gdzie *pgmname* jest nazwą programu użytkownika.

Przykładem komendy dla środowiska wielowątkowego jest:

Tabela 151. Przykład CRTPGM w środowisku wielowątkowym

Komenda	Typ programu/wyjścia
`CRTPGM PGM( pgmname ) MODULE( pgmname )` `BNDSRVPGM(QMQM/LIBMQM_R)`	Serwer lub klient dla języka C

gdzie *pgmname* jest nazwą programu użytkownika.

W poniższych tabelach przedstawiono biblioteki, które są wymagane podczas przygotowywania programów w języku C w systemie IBM i w środowisku niewątkowym oraz w środowisku wielowątkowym.

Tabela 152. Środowisko niewielowątkowe	
Zbiór biblioteki	**Typ programu/wyjścia**
LIBMQM	Serwer dla C
LIBMQIC & LIBMQM	Klient dla C

Tabela 153. Środowisko gwintowane	
Zbiór biblioteki	**Typ programu/wyjścia**
LIBMQM_R	Serwer dla C
LIBMQIC_R & LIBMQM_R	Klient dla C

IBM i *Przygotowywanie programów w języku COBOL w produkcie IBM i*

Ta sekcja zawiera informacje na temat przygotowywania programów w języku COBOL w produkcie IBM i oraz metody 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 powiązany proceduralny interfejs wywołania 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.

W celu uzyskania dostępu do funkcji MQI używana jest standardowa składnia języka COBOL CALL.

Pliki kopii w języku COBOL zawierające stałe nazwane i definicje struktur używane z interfejsem MQI są zawarte w źródłowym zbiorze fizycznym QMQM/QCBLLESRC.

W plikach kopii w języku COBOL używany jest pojedynczy znak cudzysłowu (') jako ogranicznik łańcucha. Kompilatory języka COBOL produktu IBM i zakładają, że ogranicznikiem jest znak cudzysłowu ("). Aby zapobiec generowaniu komunikatów ostrzegawczych przez kompilatory, należy określić 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 połączenia dynamicznego nie jest dostępny w produkcie IBM MQ 9.0 lub nowszym.

Aby użyć interfejsu wywołania procedury skonsolidowanej, wykonaj następujące kroki.

Procedura

1. Utwórz moduł przy użyciu kompilatora **CRTCBLMOD** , określając parametr:

   ```
   LINKLIT(*PRC)
   ```

2. Aby utworzyć obiekt programu, należy użyć komendy **CRTPGM** , podając odpowiedni parametr:

   Dla aplikacji niewielowątkowych:

   ```
   BNDSRVPGM(QMQM/AMQ0STUB)        Server for COBOL for non-threaded applications
   BNDSRVPGM(QMQM/AMQCSTUB)        Client for COBOL for non-threaded applications
   ```

   Dla 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 przy użyciu kompilatora języka ILE COBOL V4R4 i zawierającego opcję THREAD (SERIALIZE) w instrukcji PROCESS, programy w języku COBOL nie mogą używać wielowątkowych bibliotek produktu IBM MQ . Nawet jeśli program w języku COBOL został w ten sposób bezpieczny, należy zachować ostrożność podczas projektowania aplikacji, ponieważ THREAD (SERIALIZE) wymusza serializację procedur COBOL na poziomie modułu i może mieć wpływ na ogólną wydajność.

Więcej informacji na ten temat zawierają publikacje *WebSphere Development Studio: ILE COBOL Programmer's Guide* i *WebSphere Development Studio: ILE COBOL Reference* .

Więcej informacji na temat kompilowania aplikacji CICS znajduje się w publikacji *CICS for IBM i Application Programming Guide*, SC41-5454.

IBM i *Przygotowywanie programów CICS w programie IBM i*

Sekcja zawiera informacje na temat kroków wymaganych podczas przygotowywania programów CICS w programie IBM i.

Aby utworzyć program, który zawiera instrukcje EXEC CICS i wywołania MQI, wykonaj następujące kroki:

1. Jeśli to konieczne, przygotuj mapy za pomocą komendy CRTCICSMAP.
2. Przetłumaczenie komend EXEC CICS na instrukcje języka rodzimego. Użyj komendy CRTCICSC dla programu w języku C. Użyj komendy CRTCICSCBL dla programu w języku COBOL.

   Uwzględnij `CICSOPT(*NOGEN)` w komendach CRTCICSC lub CRTCICSCBL. Ta liczba zatrzymuje przetwarzanie, aby umożliwić uwzględnienie odpowiednich programów usługowych CICS i IBM MQ . Ta komenda domyślnie umieszcza kod w katalogu QTEMP/QACYCICS.
3. Skompiluj kod źródłowy za pomocą komendy CRTCMOD (dla programu C) lub komendy CRTCBLMOD (dla programu w języku COBOL).
4. Użyj komendy CRTPGM, aby połączyć skompilowany kod z odpowiednimi programami usługowym CICS i IBM MQ . Spowoduje to utworzenie programu wykonywalnego.

Poniżej przedstawiono przykład takiego kodu (kompilacja dostarczanego programu przykładowego programu 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 programie IBM i*

Jeśli używany jest produkt IBM MQ for IBM i, można zapisywać aplikacje w języku RPG.

Więcej informacji na ten temat zawiera sekcja “Kodowanie programów IBM MQ w języku RPG (tylkoIBM i)” na stronie 1165, a także informacje na temat Skorowidz programistyczny aplikacji IBM i (ILE/RPG).

IBM i *Zagadnienia związane z programowaniem języka IBM i dla*

Sekcja zawiera informacje na temat kroków wymaganych przy budowaniu aplikacji w systemie IBM i przy użyciu języka SQL.

Jeśli program zawiera instrukcje EXEC SQL i wywołania MQI, wykonaj następujące kroki:

1. Przetłumaczenie komend EXEC SQL na instrukcje języka rodzimego. Użyj komendy CRTSQLCI dla programu C. Użyj komendy CRTSQLCBLI dla programu w języku COBOL.

   Uwzględnij `OPTION(*NOGEN)` w komendzie CRTSQLCI lub CRTSQLCBLI. Ta liczba zatrzymuje przetwarzanie, aby umożliwić dołączenie odpowiednich programów usługowych produktu IBM MQ . Komenda ta domyślnie umieszcza kod w katalogu QTEMP/QSQLTEMP.

2. Skompiluj kod źródłowy za pomocą komendy CRTCMOD (dla programu C) lub komendy CRTCBLMOD (dla programu w języku COBOL).
3. Użyj komendy CRTPGM, aby połączyć skompilowany kod z odpowiednimi programami usługowym IBM MQ . Spowoduje to utworzenie programu wykonywalnego.

Poniżej znajduje się przykład takiego kodu (kompilacja programu, 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

W tej sekcji opisano dodatkowe zadania oraz zmiany w standardowych zadaniach, które należy wykonać podczas budowania aplikacji IBM MQ na potrzeby uruchamiania aplikacji Linux .

Opcje C i C++ są obsługiwane. Informacje na temat przygotowywania programów w języku C + + znajdują się w sekcji Korzystanie z języka C++.

Linux *Przygotowywanie programów w języku C w programie Linux*

Wstępnie skompilowane 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 najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Praca w normalnym środowisku. Więcej informacji na temat programowania 64-bitowych aplikacji można znaleźć w sekcji Standardy kodowania na platformach 64-bitowych.

Łączenie bibliotek

W poniższych tabelach przedstawiono biblioteki, które są wymagane podczas przygotowywania programów w języku C w systemie Linux.

- Należy połączyć programy z odpowiednią biblioteką udostępnionej przez program IBM MQ.

  W środowisku innym niż wielowątkowe należy utworzyć odsyłacz tylko do jednej z następujących bibliotek:

Zbiór biblioteki	Typ programu/wyjścia
libmqm.so	Serwer dla C
libmqic.so & libmqm.so	Klient dla C

  W środowisku wielowątkowym należy utworzyć odsyłacz tylko do jednej z następujących bibliotek:

Zbiór biblioteki	Typ programu/wyjścia
libmqm_r.so	Serwer dla C
libmqic_r.so & libmqm_r.so	Klient dla C

  Uwaga:

  1. Nie można połączyć się z więcej niż jedną biblioteką. Oznacza to, że nie można jednocześnie połączyć się z biblioteką wielowątkową i niewielowątkową.
  2. Jeśli użytkownik zapisuje usługę instalowalną (więcej informacji na ten temat znajduje się w publikacji Administrowanie), należy przejść do biblioteki produktu libmqmzf.so .

3. W przypadku tworzenia aplikacji na potrzeby zewnętrznej koordynacji za pomocą menedżera transakcji zgodnego z interfejsem XA, takiego 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) oraz biblioteki produktu `libmqz.so` w aplikacji niewielowątkowej oraz `libmqmxa_r.so` (lub `libmqmxa64_r.so`). i bibliotek produktu `libmqz_r.so` w aplikacji wielowątkowej.
4. Biblioteki produktu IBM MQ muszą być dowiązane przed 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 najwyższego poziomu, w którym zainstalowany jest 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-bitowy, Niewątkowy

```
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-bitowa, wielowątkowa

```
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-bitowy, niewątkowy

```
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-bitowa, wielowątkowa

```
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 32-bitowych programów w różnych środowiskach.

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest 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, nie wielowątkowa

```
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, nie wielowątkowa

```
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, bez wątków

```
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, nie są gwintowane

```
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 tworzenia programów 64-bitowych w różnych środowiskach.

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest 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, nie wielowątkowa

```
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, nie wielowątkowa

```
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, nie jest wielowątkowa

```
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 produkcie Linux*

Informacje na temat przygotowywania programów w języku COBOL w produkcie Linux i przygotowywania programów w języku COBOL za pomocą programu IBM COBOL for Linux on x86 and Micro Focus COBOL.

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

1. 32-bitowe podręczniki do kopiowania w języku 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 w następującym katalogu instalowane są 64-bitowe podręczniki w języku COBOL:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy64
   ```

3. W poniższych przykładach ustaw COBCPY na:

   ```
   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:

Zbiór biblioteki	Typ programu/wyjścia
libmqmcb.so	Serwer dla języka COBOL
libmqicb.so	Klient dla języka COBOL
libmqmcb_r.so	Serwer dla języka COBOL (aplikacja wielowątkowa)
libmqicb_r.so	Klient dla języka COBOL (aplikacja wielowątkowa)

Przygotowywanie programów w języku COBOL za pomocą programu IBM COBOL for Linux na platformie x86

Przykładowe programy w języku COBOL są dostarczane z produktem IBM MQ. Aby skompilować taki program, wprowadź odpowiednią komendę z następującej listy:

32-bitowa aplikacja serwera, która nie jest wielowątkowa

```
$ cob2 -o amq0put0 amq0put0.cbl -q"BINARY(BE)" -q"FLOAT(BE)" -q"UTF16(BE)"
-L MQ_INSTALLATION_PATH/lib -lmqmcb -ICOBCPY_VALUE
```

32-bitowa aplikacja kliencka, która nie jest wielowątkowa

```
$ 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 wielowątkowego

```
$ 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 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 Micro Focus COBOL

Ustaw zmienne środowiskowe przed kompilacją programu w następujący sposób:

```
export COBCPY=COBCPY_VALUE
export LIB= MQ_INSTALLATION_PATH lib:$LIB
```

Aby skompilować 32-bitowy program w języku COBOL, w którym jest obsługiwany, używając Micro Focus COBOL, wpisz:

```
$ 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, wpisz:

```
$ 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 zmiennych środowiskowych, które są potrzebne, można znaleźć w dokumentacji produktu Micro Focus COBOL.

Solaris Budowanie aplikacji proceduralnej w systemie Solaris

Te informacje opisują dodatkowe zadania oraz zmiany w standardowych zadaniach, które należy wykonać podczas budowania aplikacji produktu IBM MQ for Solaris w celu uruchomienia w ramach produktu Solaris.

Obsługiwane są języki programowania COBOL, C i C++. Informacje na temat przygotowywania programów w języku C + + znajdują się w sekcji Korzystanie z języka C++.

Oprócz kodowania wywołań MQI w kodzie źródłowym konieczne jest dodanie odpowiednich plików włączanych. Zaznajomić się z zawartością tych plików. Pełny opis znajduje się w sekcji “Pliki definicji danych produktu IBM MQ” na stronie 806 .

W tym temacie znak ukośnika odwrotnego (\) jest używany do dzielenia długich komend na więcej niż jedną linię. Nie wprowadzaj tego znaku, wpisz każdą komendę jako pojedynczą linię.

Solaris *Przygotowywanie programów w języku C w programie Solaris*

Wstępnie skompilowane programy w języku C są dostarczane w katalogu *MQ_INSTALLATION_PATH*/samp/bin .

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Więcej informacji na temat programowania 64-bitowych aplikacji można znaleźć w sekcji Standardy kodowania na platformach 64-bitowych.

Jeśli programy mają być używane na komputerze, na którym jest zainstalowany tylko IBM MQ MQI client for Solaris , należy skompilować programy w celu połączenia ich z biblioteką klienta (`-lmqic`).

Jeśli używany jest nieobsługiwany kompilator `/usr/ucb/cc`, aplikacja może pomyślnie skompilować i połączyć się pomyślnie. Jednak po uruchomieniu aplikacji kończy się ona niepowodzeniem podczas próby nawiązania połączenia z menedżerem kolejek.

Uwaga: Działanie 32-bitowych klientów SSL i TLS systemu Solaris x86 skonfigurowanych na potrzeby operacji zgodnych ze standardem FIPS 140-2 zakończy się niepowodzeniem w przypadku uruchomienia w systemie z procesorem Intel. To niepowodzenie występuje, ponieważ 32-bitowy plik biblioteki GSKit-Crypto systemu Solaris x86 zgodny ze standardem FIPS 140-2 nie jest ładowany w układzie Intel. W systemach, których to dotyczy, w dzienniku błędów klienta zgłaszany jest błąd AMQ9655. Aby rozwiązać ten problem, należy wyłączyć zgodność ze standardem FIPS 140-2 lub ponownie skompilować aplikację kliencką w formacie 64-bitowym, ponieważ problem ten nie dotyczy kodu 64-bitowego.

Łączenie bibliotek

Należy połączyć się z bibliotekami produktu IBM MQ , które są odpowiednie dla danego typu aplikacji:

Pliki w komponencie Biblioteka	Typ programu/wyjścia
libmqm.so	Serwer dla C
libmqic.so & libmqm.so	Klient dla C

Uwaga:

1. Jeśli zapisujesz usługę instalowalną (więcej informacji na ten temat zawiera Administrowanie), należy przejść do biblioteki `libmqmzf.so` .
2. W przypadku tworzenia aplikacji na potrzeby zewnętrznej koordynacji za pomocą menedżera transakcji zgodnego z interfejsem XA, takiego jak IBM TXSeries Encina lub BEA Tuxedo, należy połączyć się z `libmqmxa.so` (lub `libmqmxa64.so` , jeśli menedżer transakcji traktuje typ "long" jako 64-bitowy) i biblioteki produktu `libmqz.so` .
3. Biblioteki produktu IBM MQ muszą być dowiązane przed innymi bibliotekami produktu.

Solaris *Budowanie aplikacji na platformie Solaris x86-64*

Ten temat zawiera przykłady komend używanych do budowania programów w różnych środowiskach na platformie Solaris x86-64 .

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Aplikacja kliencka C, 32-bitowa

```
cc -xarch=386 -mt -o amqsputc_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib
-R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqic -lsocket -lnsl -ldl
```

Aplikacja kliencka C, 64-bitowe

```
cc -xarch=amd64 -mt -o amqsputc_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqic -lsocket
-lnsl -ldl
```

Aplikacja serwera C, 32-bitowa

```
cc -xarch=386 -mt -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib
-R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqm -lsocket -lnsl -ldl
```

Aplikacja serwera C, 64-bitowe

```
cc -xarch=amd64 -mt -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqm -lsocket
-lnsl -ldl
```

Aplikacja kliencka C + +, 32-bitowa

```
CC -xarch=386 -mt -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib
-R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqc23as -limqb23as -lmqic -lsocket -lnsl -ldl
```

Aplikacja kliencka C++ (64-bitowe)

```
CC -xarch=amd64 -mt -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqc23as
-limqb23as
-lmqic -lsocket -lnsl -ldl
```

Aplikacja serwera C + +, 32-bitowa

```
CC -xarch=386 -mt -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib
-R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqs23as -limqb23as -lmqm
-lsocket -lnsl -ldl
```

Aplikacja serwera C + +, 64-bitowa

```
CC -xarch=amd64 -mt -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
-limqb23as -lmqm
-lsocket -lnsl -ldl
```

Wyjście klienta C, 32-bitowe

```
cc -xarch=386 -mt -G -KPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib
-R/usr/lib/32
-lmqic -lsocket -lnsl -ldl
```

Wyjście klienta C, 64-bitowe

```
cc -xarch=amd64 -mt -G -KPIC -o /var/mqm/exits64/cliexit_64 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64
-R/usr/lib/64
-lmqic -lsocket -lnsl -ldl
```

Wyjście serwera C, 32-bitowe

```
cc -xarch=386 -mt -G -KPIC -o /var/mqm/exits/srvexit_32 srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib
-R/usr/lib/32
-lmqm -lsocket -lnsl -ldl
```

Wyjście serwera C, 64-bitowe

```
cc -xarch=amd64 -mt -G -KPIC -o /var/mqm/exits64/srvexit_64 srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64
-R/usr/lib/64
-lmqm -lsocket -lnsl -ldl
```

Budowanie aplikacji na platformie Solaris SPARC
Ten temat zawiera przykłady komend używanych do budowania programów w różnych środowiskach na platformie SPARC Solaris .

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Aplikacja kliencka C, 32-bitowa

```
cc -xarch=v8plus -mt -o amqsputc_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib
-R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqic -lsocket -lnsl -ldl
```

Aplikacja kliencka C, 64-bitowe

```
cc -xarch=v9 -mt -o amqsputc_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqic
-lsocket -lnsl -ldl
```

Aplikacja serwera C, 32-bitowa

```
cc -xarch=v8plus -mt -o amqsput_32 amqsput0.c -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib
-R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -lmqm -lsocket -lnsl -ldl
```

Aplikacja serwera C, 64-bitowe

```
cc -xarch=v9 -mt -o amqsput_64 amqsput0.c -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqm
-lsocket -lnsl -ldl
```

Aplikacja kliencka C + +, 32-bitowa

```
CC -xarch=v8plus -mt -o imqsputc_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqc23as -limqb23as
-lmqic
-lsocket -lnsl -ldl
```

Aplikacja kliencka C++ (64-bitowe)

```
CC -xarch=v9 -mt -o imqsputc_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqc23as
-limqb23as
-lmqic -lsocket -lnsl -ldl
```

Aplikacja serwera C + +, 32-bitowa

```
CC -xarch=v8plus -mt -o imqsput_32 imqsput.cpp -I MQ_INSTALLATION_PATH/inc -L
MQ_INSTALLATION_PATH/lib
-R MQ_INSTALLATION_PATH/lib -R/usr/lib/32 -limqs23as -limqb23as -lmqm
-lsocket -lnsl -ldl
```

Aplikacja serwera C + +, 64-bitowa

```
CC -xarch=v9 -mt -o imqsput_64 imqsput.cpp -I MQ_INSTALLATION_PATH/inc
```

```
-L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -limqs23as
-limqb23as -lmqm
-lsocket -lnsl -ldl
```

Wyjście klienta C, 32-bitowe

```
cc -xarch=v8plus -mt -G -KPIC -o /var/mqm/exits/cliexit_32 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib
-R/usr/lib/32
-lmqic -lsocket -lnsl -ldl
```

Wyjście klienta C, 64-bitowe

```
cc -xarch=v9 -mt -G -KPIC -o /var/mqm/exits64/cliexit_64 cliexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64
-R/usr/lib/64
-lmqic -lsocket -lnsl -ldl
```

Wyjście serwera C, 32-bitowe

```
cc -xarch=v8plus -mt -G -KPIC -o /var/mqm/exits/srvexit_32 srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib -R MQ_INSTALLATION_PATH/lib
-R/usr/lib/32
-lmqm -lsocket -lnsl -ldl
```

Wyjście serwera C, 64-bitowe

```
cc -xarch=v9 -mt -G -KPIC -o /var/mqm/exits64/srvexit_64 srvexit.c
-I MQ_INSTALLATION_PATH/inc -L MQ_INSTALLATION_PATH/lib64 -R MQ_INSTALLATION_PATH/lib64
-R/usr/lib/64
-lmqm -lsocket -lnsl -ldl
```

Solaris *Przygotowywanie programów w języku COBOL w produkcie Solaris*

Informacje na temat przygotowywania programów w języku COBOL w produkcie Solaris.

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

1. 32-bitowe podręczniki do kopiowania w języku COBOL są instalowane w następującym katalogu:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy32
   ```

   i dowiązania symboliczne są tworzone w:

   ```
   MQ_INSTALLATION_PATH/inc
   ```

2. 64-bitowe podręczniki do kopiowania w języku COBOL są instalowane w następującym katalogu:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy64
   ```

3. W poniższych przykładach ustaw COBCPY na:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy32
   ```

   dla aplikacji 32-bitowych, oraz:

   ```
   MQ_INSTALLATION_PATH/inc/cobcpy64
   ```

   dla aplikacji 64-bitowych.

Skompiluj programy za pomocą kompilatora Micro Focus. Pliki kopii, które deklarują struktury, znajdują się w programie *MQ_INSTALLATION_PATH*/inc:

```
$ export LIB= MQ_INSTALLATION_PATH/lib:$LIB
$ export COBCPY="COBCPY_VALUE"
```

Kompilowanie 32-bitowych programów:

- `$ cob32 -xv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb`

  Serwer dla języka COBOL
- `$ cob32 -xv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb`

  Klient dla języka COBOL
- `$ cob32 -xtv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqmcb_r`

  Serwer wątków dla języka COBOL
- `$ cob32 -xtv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib -lmqicb_r`

  Klient wielowątkowy dla COBOL

Kompilowanie programów 64-bitowych:

- `$ cob64 -xv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb`

  Serwer dla języka COBOL
- `$ cob64 -xv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb`

  Klient dla języka COBOL
- `$ cob64 -xtv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqmcb_r`

  Serwer wątków dla języka COBOL
- `$ cob64 -xtv amqs0put0.cbl -L MQ_INSTALLATION_PATH/lib64 -lmqicb_r`

  Klient wielowątkowy dla COBOL

gdzie *amqs0put0.cbl* jest programem przykładowym.

Należy połączyć program z jednym z następujących elementów:

- libmqmcb.so

  Serwer dla języka COBOL
- libmqicb.so

  Klient dla języka COBOL

Solaris *Przygotowywanie programów CICS w programie Solaris*

Sekcja zawiera informacje na temat przygotowywania programów CICS w programie Solaris.

Dostępny jest moduł przełącznika XA, który umożliwia połączenie produktu CICS z produktem IBM MQ:

Tabela 154. Podstawowy kod dla aplikacji produktu CICS (Solaris)

Opis	**C (źródło)**	**C (exec)**
Procedura inicjowania interfejsu XA	amqzscix.c	amqzsc- TXSeries dla Solaris

Transakcje należy zawsze łączyć z biblioteką IBM MQ libmqm.so, która jest bezpieczna dla wątku.

Więcej informacji na temat obsługi transakcji CICS można znaleźć w Administrowanie.

Solaris *Obsługa TXSeries CICS*

IBM MQ for Solaris supports TXSeries CICS using the XA interface.

Napisz programy IBM MQ , które są ładowane do tego samego regionu CICS w języku C lub COBOL. Nie można utworzyć połączenia wywołań MQI języka C i COBOL do tego samego regionu CICS . Większość wywołań MQI w drugim języku nie powiodła się z kodem przyczyny MQRC_H0BJ_ERR0R.

Przygotowywanie programów CICS w języku COBOL za pomocą języka Micro Focus COBOL

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Aby użyć programu Micro Focus COBOL, należy wykonać następujące czynności:

1. Dodaj moduł biblioteki środowiska wykonawczego produktu IBM MQ COBOL do biblioteki środowiska wykonawczego za pomocą następującej komendy:

   ```
   cicsmkcobol -L/usr/lib/dce -L MQ_INSTALLATION_PATH/lib \
               MQ_INSTALLATION_PATH/lib/libmqmcbrt.o -lmqe
   ```

   Uwaga: W przypadku produktu `cicsmkcobol`produkt IBM MQ nie umożliwia tworzenia wywołań MQI w języku programowania w języku C z poziomu aplikacji COBOL.

   Jeśli istniejące aplikacje mają takie wywołania, należy przenieść te funkcje z aplikacji w języku COBOL do własnej biblioteki, na przykład `myMQ.so`. Po przeniesieniu tych funkcji nie należy dołączać biblioteki IBM MQ `libmqmcbrt.o` podczas budowania aplikacji COBOL dla produktu CICS.

   Ponadto, jeśli aplikacja COBOL nie wykonuje żadnego wywołania MQI COBOL, nie należy łączyć `libmqmz_r` z `cicsmkcobol`.

   Spowoduje to utworzenie pliku metody języka Micro Focus COBOL i włączenie biblioteki COBOL środowiska wykonawczego produktu CICS do wywoływania produktu IBM MQ w systemach UNIX and Linux .

   Uwaga: Uruchom program `cicsmkcobol` tylko wtedy, gdy instalowany jest jeden z następujących produktów:

   - Nowa wersja lub wydanie Micro Focus COBOL
   - Nowa wersja lub wydanie produktu TXSeries for Solaris
   - Nowa wersja lub wydanie dowolnego obsługiwanego produktu bazodanowego (tylko dla transakcji w języku COBOL)
   - Nowa wersja lub wydanie produktu IBM MQ
2. Wyeksportuj następującą zmienną środowiskową:

   ```
   COBCPY= MQ_INSTALLATION_PATH/inc export COBCPY
   ```
3. Konwertuj, skompiluj i dowiązaj program wpisując:

   ```
   cicstcl -l COBOL -e yourprog.ccp
   ```

Przygotowywanie programów CICS C

Zbuduj programy CICS C, korzystając ze standardowych narzędzi CICS :

1. Wyeksportuj ***jeden*** z następujących zmiennych środowiskowych:
   - LDFLAGS = "-L *MQ_INSTALLATION_PATH* Biblioteka główna -lmqm_r" export LDFLAGS
   - USERLIB = "-L *MQ_INSTALLATION_PATH* biblioteka katalogowa -lmqm_r" export USERLIB

2. Konwertuj, skompiluj i dowiązaj program wpisując:

```
cicstcl -l C amqscic0.ccs
```

Przykładowa transakcja CICS C

Przykładowe źródło C dla transakcji CICS IBM MQ jest udostępniane przez program 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.

Windows Budowanie aplikacji proceduralnej w systemie Windows

W publikacjach systemów Windows opisano sposób budowania aplikacji wykonywalnych z napisanych programów.

W tym temacie opisano dodatkowe zadania oraz zmiany w standardowych zadaniach, które należy wykonać podczas budowania aplikacji produktu IBM MQ for Windows w celu uruchomienia w systemach Windows . Obsługiwane są następujące języki programowania: ActiveX, C, C + +, COBOL i Visual Basic. Aby uzyskać informacje na temat przygotowywania programów ActiveX , patrz Korzystanie z interfejsu Component Object Model Interface (WebSphere MQ Automation Classes for ActiveX). Informacje na temat przygotowywania programów w języku C + + znajdują się w sekcji Korzystanie z języka C++.

Zadania, które należy wykonać, aby utworzyć aplikację wykonywalną przy użyciu programu IBM MQ for Windows , różnią się w zależności od języka programowania, w którym jest zapisany kod źródłowy. Oprócz kodowania wywołań MQI w kodzie źródłowym, należy dodać odpowiednie instrukcje języka, aby zawierały pliki włączanych produktu IBM MQ for Windows dla używanego języka. Zaznajomić się z zawartością tych plików. Pełny opis znajduje się w sekcji “Pliki definicji danych produktu IBM MQ” na stronie 806 .

Windows *Budowanie aplikacji 64-bitowych w systemie Windows*

Zarówno 32-bitowe, jak i 64-bitowe aplikacje są obsługiwane w systemie IBM MQ for Windows. Pliki wykonywalne i pliki biblioteki produktu IBM MQ są dostarczane zarówno w postaci 32-bitowej, jak i 64-bitowej, w zależności od aplikacji, z którą pracuje użytkownik.

Pliki wykonywalne i biblioteki

Zarówno 32-bitowe, jak i 64-bitowe wersje bibliotek produktu IBM MQ są dostarczane w następujących miejscach:

Tabela 155. Położenie bibliotek produktu 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 najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Aplikacje 32-bitowe nadal działają normalnie po migracji. Pliki 32-bitowe istnieją w tym samym katalogu, co w poprzednich wersjach produktu.

Aby utworzyć wersję 64-bitową, należy upewnić się, że środowisko jest skonfigurowane do korzystania z plików biblioteki w programie *MQ_INSTALLATION_PATH* \Tools\Lib64. Upewnij się, że zmienna środowiskowa LIB nie jest ustawiona do wyszukiwania w folderze, w którym znajdują się biblioteki 32-bitowe.

Windows *Przygotowywanie programów w języku C w programie Windows*

Praca w typowym środowisku Windows ; IBM MQ for Windows nie wymaga niczego specjalnego.

Więcej informacji na temat programowania aplikacji 64-bitowych zawiera sekcja Standardy kodowania na 64-bitowych platformach.

- Powiąż programy z odpowiednimi bibliotekami udostępnionym przez produkt IBM MQ:

Zbiór 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 C
MQ_INSTALLATION_PATH\Tools\Lib\mqicxa.lib	klient dla 32-bitowego C z koordynacją transakcji
MQ_INSTALLATION_PATH\Tools\Lib64\mqm.lib	serwer dla 64-bitowego C
MQ_INSTALLATION_PATH\Tools\Lib64\mqic.lib	klient dla 64-bitowego C
MQ_INSTALLATION_PATH\Tools\Lib64\mqicxa.lib	klient dla 64-bitowego C z koordynacją transakcji

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest 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:

- Jeśli użytkownik zapisuje usługę instalowalną (więcej informacji na ten temat znajduje się w publikacji Administrowanie), należy utworzyć odsyłacz do biblioteki mqmzf.lib .
- W przypadku tworzenia aplikacji na potrzeby zewnętrznej koordynacji za pomocą menedżera transakcji zgodnego z interfejsem XA, takiego jak IBM TXSeries Encina, lub BEA Tuxedo, należy utworzyć dowiązanie do biblioteki mqmxa.lib lub mqmxa.lib .
- Jeśli użytkownik zapisuje wyjście produktu CICS , należy utworzyć odsyłacz do biblioteki mqmcics4.lib .

  - Biblioteki produktu IBM MQ muszą być dowiązane przed innymi bibliotekami produktu.
- Biblioteki DLL muszą znajdować się w podanej ścieżce (PATH).
- Jeśli w miarę możliwości używane są małe litery, można przejść z IBM MQ for Windows do IBM MQ w systemach UNIX and Linux , gdzie konieczne jest użycie małych liter.

Przygotowywanie programów CICS i serwera transakcji

Przykładowe źródło C dla transakcji CICS IBM MQ jest udostępniane przez program AMQSCIC0.CCS. Budujesz go przy użyciu standardowych narzędzi CICS . Na przykład dla TXSeries dla Windows 2000:

1. Ustaw zmienną środowiskową (wpisz 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 dowiązaj przykładowy program:

```
cicstcl -l IBMC amqscic0.ccs
```

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Jest to opisane w publikacji *Transaction Server for Windows NT Application Programming Guide (CICS) V4*.

Więcej informacji na temat obsługi transakcji CICS można znaleźć w Administrowanie.

Windows Przygotowywanie programów w języku COBOL w produkcie Windows

Ta sekcja zawiera informacje na temat przygotowywania programów w języku COBOL w produkcie Windowsoraz przygotowywania programów CICS i serwera transakcji.

1. 32-bitowe podręczniki do kopiowania języka COBOL są instalowane w następującym katalogu: *MQ_INSTALLATION_PATH* \Tools\cobol\CopyBook.
2. 64-bitowe podręczniki kopii języka COBOL są instalowane w następującym katalogu: *MQ_INSTALLATION_PATH* \Tools\cobol\CopyBook64
3. W poniższych przykładach ustaw CopyBook na:

```
CopyBook
```

   dla aplikacji 32-bitowych, oraz:

```
CopyBook64
```

   dla aplikacji 64-bitowych.

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Aby przygotować programy w języku COBOL w systemach Windows , należy połączyć program z jedną z następujących bibliotek udostępnianych przez produkt IBM MQ:

Zbiór biblioteki	Typ programu lub wyjścia
MQ_INSTALLATION_PATH \Tools\Lib\mqmcb	32-bitowy serwer dla Micro Focus COBOL

Zbiór biblioteki	Typ programu lub wyjścia
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	Klient 64-bitowy dla Micro Focus COBOL

W przypadku uruchamiania programu w środowisku klienta MQI upewnij się, że biblioteka DOSCALLS jest wyświetlana przed dowolną biblioteką języka COBOL lub IBM MQ .

Przygotowywanie programów w języku COBOL przy użyciu Micro Focus COBOL

Należy zrelować wszystkie istniejące 32-bitowe programy IBM MQ Micro Focus COBOL za pomocą `mqmcb.lib` lub `mqiccb.lib`, a nie bibliotek `mqmcbb` i `mqiccbb` .

Aby skompilować, na przykład, przykładowy program `amq0put0`, używając Micro Focus COBOL:

1. Ustaw zmienną środowiskową COBCPY tak, aby wskazywała na struktury copybook w języku COBOL produktu IBM MQ (wprowadź następujący kod w jednym wierszu):

   ```
   set COBCPY= MQ_INSTALLATION_PATH\
   Tools\Cobol\Copybook
   ```

2. Skompiluj program, aby nadać mu plik wynikowy:

   ```
   cobol amq0put0 LITLINK
   ```

3. Dowiąże plik wynikowy do systemu wykonawczego.
   - Ustaw zmienną środowiskową LIB tak, aby wskazywała na biblioteki COBOL kompilatora.
   - Odsyłacz do pliku obiektu, który ma być używany na serwerze IBM MQ :

     ```
     cbllink amq0put0.obj mqmcb.lib
     ```

   - Lub dowiązaj plik obiektu do użycia na kliencie IBM MQ :

     ```
     cbllink amq0put0.obj mqiccb.lib
     ```

Przygotowywanie programów CICS i serwera transakcji

Aby skompilować i połączyć program TXSeries dla programu Windows NT, program V5.1 za pomocą programu IBM VisualAge COBOL:

1. Ustaw zmienną środowiskową (wpisz 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. Konwertuj, skompiluj i dowiązaj swój program:

   ```
   cicstcl -l IBMCOB myprog.ccp
   ```

Jest to opisane w podręczniku *Transaction Server for Windows NT, V4 Application Programming* Guide.

Aby skompilować i połączyć program CICS dla programu Windows V5 za pomocą programu 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 programie Windows*

Informacje do rozważenia podczas korzystania z programów Microsoft Visual Basic w systemie Windows.

W produkcie IBM MQ 9.0obsługa produktu IBM MQ dla produktu Microsoft Visual Basic 6.0 jest nieaktualna. Zalecaną technologią zastępczą są klasy IBM MQ dla .NET. Więcej informacji na ten temat zawiera sekcja Tworzenie aplikacji .NET.

Uwaga: Wersje 64-bitowe plików modułu Visual Basic nie są dostarczane.

Aby przygotować programy Visual Basic w systemie Windows:

1. Utwórz nowy projekt.
2. Dodaj dostarczony plik modułu CMQB.BAS, do projektu.
3. Dodaj inne dostarczone pliki modułów, jeśli są one potrzebne:
   - CMQBB.BAS: Obsługa MQAI
   - CMQCFB.BAS: Obsługa PCF
   - CMQXB.BAS: Obsługa wyjść kanału
   - CMQPSB.BAS: Publikowanie/subskrypcja

Informacje na temat używania wywołania MQCONNXAny z poziomu produktu Visual Basiczawiera sekcja “Kodowanie w Visual Basic” na stronie 1160 .

Wywołaj procedurę MQ_SETDEFAULTS przed wywołaniem dowolnego wywołania MQI w kodzie projektu. Ta procedura służy do konfigurowania struktur domyślnych wymaganych przez wywołania MQI.

Przed skompilowaniem lub uruchomieniem projektu określ, czy tworzony jest serwer lub klient IBM MQ , 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.
4. W polu Argumenty kompilacji warunkowej wprowadź tę wartość dla serwera:

```
MqType=1
```

lub w przypadku klienta:

```
MqType=2
```

Pojęcia pokrewne

“Kodowanie w Visual Basic” na stronie 1160
Informacje, które należy wziąć pod uwagę podczas kodowania programów IBM MQ w programie Microsoft Visual Basic. Produkt Visual Basic jest obsługiwany tylko w systemie Windows.

Odsyłacze pokrewne

“Łączenie aplikacji produktu Visual Basic z kodem produktu IBM MQ MQI client” na stronie 1016
Aplikacje produktu Microsoft Visual Basic można połączyć z kodem IBM MQ MQI client w systemie Windows.

Windows *Wyjście zabezpieczeń SSPI*

Produkt 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 udostępnia uwierzytelnianie dla kanałów produktu IBM MQ przy użyciu interfejsu SSPI (Security Services Programming Interface). Interfejs SSPI udostępnia zintegrowane zabezpieczenia systemów Windows .

Pakiety zabezpieczeń są ładowane z pliku security.dll lub z pliku secur32.dll. Te biblioteki DLL są dostarczane wraz z systemem operacyjnym.

Uwierzytelnianie jednokierunkowe jest udostępniane za pomocą usług uwierzytelniania NTLM. Uwierzytelnianie dwukierunkowe jest udostępniane za pomocą usług uwierzytelniania Kerberos .

Program obsługi wyjścia zabezpieczeń jest dostarczany w formacie źródłowym i obiektowym. Można użyć kodu wynikowego, który jest, lub użyć kodu źródłowego jako punktu wyjścia do tworzenia własnych programów obsługi wyjścia użytkownika.

Patrz także “Korzystanie z wyjścia zabezpieczeń SSPI w systemie Windows” na stronie 1246.

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 do kontroli po nawiązaniu sesji MCA, jest znany jako *inicjator 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 156. 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**

  W ten sposób używane są usługi uwierzytelniania NTLM, które zapewniają uwierzytelnianie w jedną stronę. NTLM umożliwia serwerom sprawdzanie tożsamości swoich klientów. Nie pozwala ona klientom

zweryfikować tożsamości serwera lub jednego serwera w celu zweryfikowania tożsamości innego serwera. Uwierzytelnianie NTLM zostało zaprojektowane z myślą o środowisku sieciowym, w którym zakłada się, że serwery są prawdziwe.

- **SCY_KERBEROS**

  Korzysta z usług uwierzytelniania wzajemnego Kerberos . Protokół Kerberos nie zakłada, że serwery w środowisku sieciowym są prawdziwe. Strony 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, co robią programy obsługi wyjścia kanału SSPI.

Dostarczone programy obsługi wyjścia kanału udostępniają jednokierunkowe lub dwukierunkowe (wzajemne) uwierzytelnianie systemu partnerskiego w momencie tworzenia sesji. Dla konkretnego kanału, każdy program obsługi wyjścia ma powiązaną *nazwę użytkownika* (podobną do ID użytkownika, patrz “Sterowanie dostępem do produktu IBM MQ i nazwy użytkowników produktu Windows” na stronie 1128). Połączenie między dwoma programami obsługi wyjścia jest powiązaniem między tymi dwoma zleceniodawców.

Po nawiązaniu sesji bazowej zostanie nawiązane bezpieczne połączenie między dwoma programami obsługi wyjścia zabezpieczeń (jedną dla wysyłającego agenta MCA i jedną dla odbierającego agenta MCA). Sekwencja operacji jest następująca:

1. Każdy program jest powiązany z konkretnym zleceniodawca, na przykład w wyniku jawnej operacji logowania.
2. Inicjator kontekstu żąda bezpiecznego połączenia z partnerem z pakietu zabezpieczeń (dla Kerberos, nazwanego partnera) i odbiera token (o nazwie token1). Token jest wysyłany, przy użyciu bazowej sesji, która jest już ustanowiona, do programu partnerskiego.
3. Program partnerski (akceptor kontekstu) przekazuje element token1 do pakietu zabezpieczeń, który sprawdza, czy inicjator kontekstu jest autentyczny. W przypadku NTLM połączenie jest teraz nawiązane.
4. W przypadku wyjścia zabezpieczeń dostarczonego przez protokół Kerberos(w przypadku uwierzytelniania wzajemnego) pakiet zabezpieczeń generuje również drugi znacznik (o nazwie token2), który akceptor kontekstu zwraca do inicjatora kontekstu przy użyciu sesji bazowej.
5. Inicjator kontekstu korzysta z token2 w celu sprawdzenia, czy akceptor kontekstu jest autentyczny.
6. Na tym etapie, jeśli obie aplikacje są zadowolone z autentyczności tokenu partnera, nawiąże się bezpieczne (uwierzytelnione) połączenie.

Sterowanie dostępem do produktu IBM MQ i nazwy użytkowników produktu Windows

Kontrola dostępu, którą udostępnia program IBM MQ , jest oparta na użytkowniku i grupie. Uwierzytelnianie, które udostępnia produkt Windows , jest oparte na zasadach podstawowych, takich jak nazwa użytkownika i nazwa servicePrincipal(SPN). W przypadku nazwy servicePrincipalmoże istnieć wiele z nich powiązanych z jednym użytkownikiem.

Wyjście zabezpieczeń SSPI używa odpowiednich nazw użytkowników produktu Windows do uwierzytelniania. Jeśli uwierzytelnianie produktu Windows zakończy się pomyślnie, wyjście przekazuje identyfikator użytkownika powiązany z elementem głównym Windows do IBM MQ w celu kontroli dostępu.

Nazwy użytkowników produktu 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 jednym ze sposobów, nazwa użytkownika powiązana z akceptorem kontekstu jest nieistotna.
- Dla uwierzytelniania Kerberos , w kanałach CLNTCONN, nazwą użytkownika Windows jest ID użytkownika powiązany z uruchomionym procesem. W przeciwnym razie nazwą użytkownika Windows jest nazwa servicePrincipal, która jest tworzona przez dodanie następującego przedrostka do nazwy QueueManager.

```
ibmMQSeries/
```

z/OS Budowanie aplikacji proceduralnej w systemie z/OS

Publikacje CICS, IMSi z/OS opisują sposoby budowania aplikacji, które działają w tych środowiskach.

W tej kolekcji tematów opisano dodatkowe zadania oraz zmiany w standardowych zadaniach, które należy wykonać podczas budowania aplikacji produktu IBM MQ for z/OS dla tych środowisk. Obsługiwane są języki programowania COBOL, C, C + +, Assembler i PL/I. (Informacje na temat tworzenia aplikacji C++ zawiera sekcja Korzystanie z języka C++).

Zadania, które należy wykonać, aby utworzyć wykonywalną aplikację 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 programu IBM MQ for z/OS dla używanego języka. Zaznajomić się z zawartością tych plików. Pełny opis znajduje się w sekcji “Pliki definicji danych produktu IBM MQ” na stronie 806 .

Uwaga

Nazwa **thlqual** jest kwalifikatorem wysokiego poziomu dla biblioteki instalacji w systemie z/OS.

z/OS *Przygotowanie programu do uruchomienia*

Po napisaniu programu dla aplikacji IBM MQ w celu utworzenia aplikacji wykonywalnej należy skompilować lub złożyć plik, a następnie utworzyć odsyłacz do edycji wynikowego kodu wynikowego z programem pośredniczonym, który produkt IBM MQ for z/OS udostępnia dla każdego obsługiwanego środowiska.

Sposób przygotowania programu zależy zarówno od środowiska (zadania wsadowe, CICS, IMS(BMP lub MPP), Linux lub UNIX usług systemowych), w których działa aplikacja, jak i od struktury zestawów danych w instalacji produktu z/OS .

“Dynamiczne wywoływanie kodu pośredniczącego produktu IBM MQ” na stronie 1136 opisuje alternatywną metodę wykonywania wywołań MQI w programach, tak aby nie trzeba tworzyć odsyłaczy do edycji kodu pośredniczącego produktu IBM MQ . Ta metoda nie jest dostępna dla wszystkich języków i środowisk.

Nie należy tworzyć odsyłaczy do edycji wyższego poziomu programu pośredniczącego niż wersja produktu IBM MQ for z/OS , na którym jest uruchomiony program. Na przykład program działający w systemie MQSeries for OS/390, V5.2 , nie może być edytowany dowiązaniem z programem pośredniczonym dostarczonym z programem IBM MQ for z/OS V7.

z/OS *Budowanie aplikacji w wersji 64-bitowej C*

W produkcie z/OSaplikacje 64-bitowe są budowane za pomocą kompilatora LP64 i opcji konsolidatora. Plik nagłówkowy IBM MQ for z/OS *cmqc.h* rozpoznaje, kiedy ta opcja jest dostępna dla kompilatora, a także generuje typy danych i struktury IBM MQ odpowiednie dla operacji 64-bitowej.

Kod C zbudowany przy użyciu tej opcji musi być zbudowany w taki sposób, aby używać bibliotek DLL (DLL), które są odpowiednie dla wymaganej semantycznej koordynacji. Powiązanie skompilowanego kodu

z odpowiednim pokładem bocznym zdefiniowanym w polu Nazwa pokładu bocznego wymagana dla każdej semantyki koordynacyjnej pokazuje wymaganą bibliotekę DLL.

Tabela 157. Nazwa pokładu bocznego wymagana dla każdej semantycznej koordynacji

spójnik	**Nazwa pokładu bocznego**
Zatwierdzanie jednofazowe MQI	CSQBMQ2X
Zatwierdzanie dwufazowe z koordynacją RRS za pomocą komend RRS	CSQBRR2X
Zatwierdzanie dwufazowe z koordynacją RRS za pomocą komend MQI	CSQBRI2X

Użyj procedury EDCQCB JCL dostarczonej wraz z produktem *z/OS XL C/C++*, aby zbudować program IBM MQ zatwierdzania jednofazowego 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 skoordynowany program RRS w programie z/OS Unix System Services, skompiluj i dowiązaj 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 produktu z/OS*

Sekcja zawiera informacje na temat budowania aplikacji wsadowych produktu z/OS oraz kroków, które należy wykonać w celu rozważenia wykonania tego zadania.

Aby zbudować aplikację dla produktu IBM MQ for z/OS , która działa w ramach zadania wsadowego z/OS , należy utworzyć język sterowania zadaniami (JCL), który wykonuje następujące zadania:

1. Skompiluj (lub zmontuj) program, aby utworzyć kod obiektu. Kod JCL dla kompilacji musi zawierać instrukcje SYSLIB, które sprawiają, że pliki definicji danych produktu są dostępne dla kompilatora. Definicje danych są dostarczane w następujących bibliotekach produktu IBM MQ for z/OS :
   - Dla języka COBOL: **thlqual**.SCSQCOBC
   - Dla języka asemblera: **thlqual**.SCSQMACS
   - Dla C, **thlqual**.SCSQC370
   - W przypadku języka PL/I, **thlqual**.SCSQPLIC
2. W przypadku aplikacji C należy wcześniej utworzyć powiązanie z kodem obiektu utworzonym w kroku “1” na stronie 1130.
3. W przypadku aplikacji PL/I należy użyć opcji kompilatora EXTRN (SHORT).
4. Odsyłacz-edytuj kod obiektu utworzony w kroku “1” na stronie 1130 (lub krok “2” na stronie 1130 dla aplikacji C), aby utworzyć moduł ładowalny. Po kliknięciu odsyłacza-edytuj kod, należy dołączyć do niego jeden z programów wsadowych IBM MQ for z/OS (CSQBSTUB lub jeden z programów pośredniczących RRS: CSQBRRSI lub CSQBRSTB).

   SZKIEŁ_BD
   zatwierdzanie jednofazowe udostępniane przez produkt IBM MQ for z/OS

CSQBRRSI
zatwierdzanie dwufazowe udostępniane przez usługi RRS przy użyciu interfejsu MQI

CSQBRSTB
zatwierdzanie dwufazowe udostępniane przez RRS bezpośrednio

Uwagi:

a. Jeśli używany jest kod CSQBRSTB, należy również utworzyć dowiązanie do edycji aplikacji z systemem ATRSCSS z poziomu SYS1.CSSLIB. Rysunek 118 na stronie 1131 i Rysunek 119 na stronie 1131 przedstawiają fragmenty JCL do wykonania tego zadania. Kody pośredniczące są niezależne od języka i są dostarczane w bibliotece **thlqual**.SCSQLOAD.

b. Jeśli aplikacja jest uruchamiana w środowisku Language Environment, należy upewnić się, że zamiast tego opisano odsyłacz do biblioteki DLL środowiska językowego, a nie w sposób opisany w sekcji “Budowanie aplikacji wsadowych produktu z/OS przy użyciu środowiska językowego” na stronie 1132.

5. Zapisz moduł ładujący w bibliotece ładowania aplikacji.

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING BATCH STUB
//*
//CSQSTUB   DD  DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//*
⋮
//SYSIN     DD *
  INCLUDE CSQSTUB(CSQBSTUB)
⋮
/*
```

Rysunek 118. Fragmenty kodu JCL do połączenia-edycja modułu obiektu w środowisku wsadowym za pomocą 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 119. Fragmenty kodu JCL do połączenia-edycja modułu obiektu w środowisku wsadowym za pomocą zatwierdzania dwufazowego

Aby uruchomić program wsadowy lub RRS, należy dołączyć biblioteki **thlqual**.SCSQAUTH i **thlqual**.SCSQLOAD w konkatenacji zestawu danych STEPLIB lub JOBLIB.

Aby uruchomić program TSO, należy uwzględnić biblioteki **thlqual**.SCSQAUTH i **thlqual**.SCSQLOAD w bibliotece STEPLIB, które są używane przez sesję TSO.

Aby uruchomić program wsadowy programu UNIX System Services z poziomu powłoki usług systemu UNIX , należy dodać 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 produktu z/OS przy użyciu środowiska językowego*

Produkt IBM MQ for z/OS udostępnia zestaw dynamicznych bibliotek połączeń (DLL), które muszą być używane podczas tworzenia odsyłaczy do edycji aplikacji.

Istnieją dwa warianty bibliotek, które umożliwiają aplikacji korzystanie z jednego z następujących interfejsów wywołującej:

- 31-bitowy interfejs językowy środowiska językowego.
- 31-bitowy interfejs wywołujący XPLINK. z/OS XPLINK to konwencja o wysokiej wydajności, dostępna dla aplikacji w języku C.

Aby można było korzystać z bibliotek DLL, aplikacja jest powiązana lub powiązana z nazwą *sidedecks*, a nie z kodami pośrednicznymi udostępnianym we wcześniejszych wersjach. Komendy sidedecks znajdują się w bibliotece SCSQDEFS (zamiast biblioteki SCSQLOAD).

Tabela 158. Warianty bibliotek połączeń dynamicznych

Zatwierdzanie	**31-bitowa biblioteka DLL środowiska językowego**	**31-bit XPLINK DLL**	**Równoważna nazwa kodu pośredniczącego**
1 faza zatwierdzania bibliotek MQI	CSQBMQ1	CSQBMQ1X	SZKIEŁ_BD
Zatwierdzanie dwufazowe z koordynacją RRS za pomocą czasowników sterowanych transakcjami RRS	CSQBRR1	CSQBRR1X	CSQBRSTB
Zatwierdzanie dwufazowe z koordynacją RRS za pomocą czasowników sterowanych transakcjami MQI	CSQBRI1	CSQBRI1X	CSQBRRSI

Uwaga: Wszystkie sidedecks zawierają definicję punktu wejścia konwersji danych, MQXCNVC, poprzednio rozstrzygnięte przez dołączenie CSQASTUB.

Często występujące problemy:

- Jeśli aplikacja korzysta z asynchronicznego wykorzystania komunikatów (wywołania MQCB, MQCTL lub MQSUB), to w protokole zadania pojawia się następujący komunikat, a poprzedni interfejs DLL nie jest używany:

  ```
  CSQB001E Programy środowiska języka działające w trybie wsadowym z/OS lub USS muszą używać
  interfejsu DLL do IBM MQ
  ```

  Rozwiązanie: Odbuduj aplikację, używając funkcji sidedecks zamiast kodów pośredniczących, tak jak poprzednio.

- W czasie budowania programu pojawia się następujący komunikat

  ```
  IEW2469E Atrybuty odwołania do MQAPI-NAME z sekcji kod-użytkownika nie są zgodne z atrybutami
  elementu.
  Symbol elementu docelowego
  ```

  Przyczyna: Oznacza to, że program XPLINK został skompilowany z produktem V701 (lub nowszym) wersją produktu cmqc.h, ale nie są one wiążące dla sidedecks.

  Rozwiązanie: Należy zmienić plik budowania programu, tak aby został powiązany z odpowiednim rozszerzeniem z SCSQDEFS zamiast kodu pośredniczącego z SCSQLOAD.

Poniższy przykładowy skrypt JCL pokazuje, w jaki sposób można skompilować i dowiązywać-edytować program C tak, aby używać 31-bitowego interfejsu DLL środowiska językowego języka DLL:

```
//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** . Edycja odsyłacza korzysta z opcji **DYNAM=DLL** , a odwołania do biblioteki **CSQBMQ1** .

Poniższy przykładowy skrypt JCL pokazuje, w jaki sposób można skompilować i dowiązywać-edytować program C, aby używać 31-bitowego interfejsu wywołującego 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** . Edycja odsyłacza korzysta z 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 NIEROZSTRZYGNIĘTE, są wskazaniem, że należy sprawdzić, czy wszystkie programy zostały skompilowane z opcją DLL.

z/OS *Budowanie aplikacji CICS w produkcie z/OS*

Te informacje są używane podczas budowania aplikacji produktu CICS w produkcie z/OS.

Aby zbudować aplikację dla produktu IBM MQ for z/OS , która działa w ramach produktu CICS, należy wykonać następujące czynności:

- Przetłumaczenie komend produktu CICS w programie na język, w którym znajduje się pozostała część programu użytkownika.
- Skompiluj lub zmontuj dane wyjściowe z translatora w celu utworzenia kodu obiektu.
  - W przypadku programów PL/I należy użyć opcji kompilatora EXTRN (SHORT).

  - W przypadku aplikacji C, jeśli aplikacja nie korzysta z produktu XPLINK, należy użyć opcji DEFINE kompilatora (MQ_OS_LINKAGE=1).
- Odsyłacz-edytuj kod obiektu, aby utworzyć moduł ładowalny.

Produkt CICS udostępnia procedurę wykonywania tych kroków w kolejności dla każdego z obsługiwanych języków programowania.

- W przypadku produktu CICS Transaction Server for z/OSw *Podręcznik definicji systemu CICS Transaction Server for z/OS* opisano sposób korzystania z tych procedur, a program *Podręcznik programowania aplikacji CICS/ESA* udostępnia więcej informacji na temat procesu translacji.

Należy uwzględnić:

- W instrukcji SYSLIB na etapie kompilacji (lub zespołu) instrukcji, które udostępniają kompilatorowi pliki definicji danych produktu. Definicje danych są dostarczane w następujących bibliotekach produktu IBM MQ for z/OS :
  - Dla języka COBOL: **thlqual**.SCSQCOBC
  - Dla języka asemblera: **thlqual**.SCSQMACS
  - Dla C, **thlqual**.SCSQC370
  - W przypadku języka PL/I, **thlqual**.SCSQPLIC
- W edytorze definicji połączeń JCL program IBM MQ for z/OS CICS -kod pośredniczący (CSQCSTUB). Program Rysunek 120 na stronie 1134 wyświetla fragmenty kodu JCL w celu wykonania tego zadania. 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 120. Fragmenty kodu JCL do połączenia-edycja modułu obiektu w środowisku produktu CICS

- W przypadku wersji CICS nowszych niż CICS TS 3.2lub, jeśli mają być używane interfejsy API właściwości komunikatu produktu IBM MQ , lub funkcje API IBM MQ API MQCB, MQCTL, MQSTAT, MQSUB lub MQSUBR, należy przeprowadzić połączenie kodu obiektu z dostarczonym kodem pośredniczonym produktu CICS , DFHMQSTB, a nie dostarczonym przez produkt IBM MQ CSQCSTUB. Więcej informacji na temat tworzenia programów IBM MQ dla produktu CICSzawiera sekcja Program pośredniczący interfejsu API w celu uzyskania dostępu do wywołań MQI produktu IBM MQ w dokumentacji produktu CICS .

Po wykonaniu tych czynności należy zapisać moduł ładujący w bibliotece ładowania aplikacji i zdefiniować program CICS w zwykły sposób.

Przed uruchomieniem programu CICS administrator systemu musi zdefiniować go do programu CICS jako program i transakcję IBM MQ , a następnie uruchomić go w typowy sposób.

z/OS *Budowanie aplikacji IMS (BMP lub MPP)*

Te informacje są używane podczas budowania aplikacji IMS (BMP lub MPP).

Jeśli budujesz wsadowe programy DL/I, zapoznaj się z “Budowanie aplikacji wsadowych produktu z/OS” na stronie 1130. Aby zbudować inne aplikacje, które działają pod kontrolą produktu IMS (jako BMP lub MPP), utwórz JCL wykonuj następujące zadania:

1. Skompiluj (lub zmontuj) program, aby utworzyć kod obiektu. Kod JCL dla kompilacji musi zawierać instrukcje SYSLIB, które sprawiają, że pliki definicji danych produktu są dostępne dla kompilatora. Definicje danych są dostarczane w następujących bibliotekach produktu IBM MQ for z/OS :

- W języku COBOL: **thlqual**.SCSQCOBC
- Dla języka asemblera: **thlqual**.SCSQMACS
- Dla C, **thlqual**.SCSQC370
- W przypadku języka PL/I, **thlqual**.SCSQPLIC

2. W przypadku aplikacji C należy wcześniej utworzyć dowiązanie wstępne do modułu obiektu utworzonego w kroku “1” na stronie 1134.
3. W przypadku programów PL/I należy użyć opcji kompilatora EXTRN (SHORT).
4. W przypadku aplikacji C, jeśli aplikacja nie używa produktu XPLINK, należy użyć opcji kompilatora DEFINE (MQ_OS_LINKAGE=1).
5. Odsyłacz-edytuj kod obiektu utworzony w kroku “1” na stronie 1134 (lub krok “2” na stronie 1135 dla aplikacji C/370), aby utworzyć moduł ładowania:
   a. Dołącz moduł interfejsu językowego IMS (DFSLI000).
   b. Uwzględnij program pośredniczący IBM MQ for z/OS IMS (CSQQSTUB). Program Rysunek 121 na stronie 1135 wyświetla fragmenty kodu JCL do wykonania tego zadania. 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 edytor powiązań rozstrzyganie odwołań do tabeli CSQQSTUB, chyba że planowane jest korzystanie z łącznika dynamicznego zgodnie z opisem w sekcji “Dynamiczne wywoływanie kodu pośredniczącego produktu IBM MQ” na stronie 1136.
6. Zapisz moduł ładujący w bibliotece ładowania aplikacji.

```
⋮
//*
//* WEBSPHERE MQ FOR Z/OS LIBRARY CONTAINING IMS STUB
//*
//CSQSTUB   DD  DSN=thlqual.SCSQLOAD,DISP=SHR
//*
⋮
//LKED.SYSIN DD *
  INCLUDE CSQSTUB(CSQQSTUB)
  ⋮
/*
```

Rysunek 121. Fragmenty kodu JCL do połączenia-edycja modułu obiektu w środowisku produktu IMS

Przed uruchomieniem programu IMS administrator systemu musi go zdefiniować jako IMS jako program i transakcję IBM MQ : następnie można uruchomić go w typowy sposób.

z/OS *Budowanie aplikacji usług systemowych z/OS UNIX*

Te informacje są używane podczas budowania aplikacji z/OS UNIX System Services.

Aby zbudować aplikację C dla produktu IBM MQ for z/OS , która działa w ramach programu UNIX System Services, skompiluj aplikację i powiąż ją w następujący sposób:

```
cc -o mqsamp -W c,DLL -I "//' thlqual.SCSQC370'" mqsamp.c "//' thlqual.SCSQDEFS(CSQBMQ1)'"
```

gdzie **thlqual** jest kwalifikatorem wysokiego poziomu używanym przez instalację.

Aby uruchomić program w języku C, należy dodać następujący plik do pliku `.profile` . Ten plik powinien znajdować się w katalogu głównym:

```
STEPLIB= thlqual.SCSQANLE:thlqual.SCSQAUTH: STEPLIB
```

Należy pamiętać, że konieczne jest wyjście z programu UNIX System Services i ponowne wprowadzenie opcji UNIX System Services, aby zmiana została rozpoznana.

Jeśli chcesz uruchomić wiele powłok, dodaj słowo eksport na początku wiersza, to znaczy:

```
export STEPLIB= thlqual.SCSQANLE:thlqual.SCSQAUTH: STEPLIB
```

Po pomyślnym zakończeniu połączenia można połączyć CSQBSTUB i wywołać wywołania IBM MQ .

“Dynamiczne wywoływanie kodu pośredniczącego produktu IBM MQ” na stronie 1136 opisuje alternatywną metodę wykonywania wywołań MQI w programach, tak aby nie trzeba tworzyć odsyłaczy do edycji kodu pośredniczącego produktu IBM MQ . Ta metoda nie jest dostępna dla wszystkich języków i środowisk.

Nie należy tworzyć odsyłaczy do edycji wyższego poziomu programu pośredniczącego niż wersja produktu IBM MQ for z/OS , na którym jest uruchomiony program. Na przykład program działający w systemie IBM WebSphere MQ for z/OS 7.1 nie może być edytowany przez odsyłacz z programem pośredniczonym dostarczonym z produktem IBM MQ for z/OS 8.0.

z/OS *Dynamiczne wywoływanie kodu pośredniczącego produktu IBM MQ*

Zamiast tworzyć odsyłacze do edycji programu pośredniczącego produktu IBM MQ przy użyciu kodu obiektu, można dynamicznie wywoływać kod pośredniczący z poziomu programu.

Można to zrobić w środowiskach wsadowych, IMSi CICS . To narzędzie nie jest obsługiwane w środowisku RRS. Jeśli program użytkowy używa usługi RRS do koordynowania aktualizacji, patrz “Uwagi dotyczące RRS” na stronie 1140.

Metoda ta jest jednak następująca:

- Zwiększa złożoność programów
- Zwiększa pamięć masową wymaganą przez programy w czasie wykonywania.
- Redukuje wydajność programów
- Oznacza, że nie można używać tych samych programów w innych środowiskach.

Jeśli kod pośredniczący zostanie dynamicznie wywołany, odpowiedni program pośredniczący i jego aliasy muszą być dostępne w czasie wykonywania. Aby to upewnić się, należy dołączyć zestaw danych SCSQLOAD zestawu danych IBM MQ for z/OS :

- W przypadku zadań wsadowych i IMSw konkatenacji STEPLIB zadania JCL.
- W przypadku produktu CICSw konkatenacji CICS DFHRPL.

W przypadku produktu IMSupewnij się, że biblioteka zawierająca dynamiczny kod pośredniczący (zbudowana zgodnie z opisem w informacjach dotyczących instalowania adaptera IMS w sekcji Konfigurowanie adaptera produktu IMS) wyprzedza zestaw danych SCSQLOAD w konkatenacji STEPLIB regionu JCL.

Nazwy pokazywane w programie Tabela 159 na stronie 1136 są używane podczas dynamicznego wywoływania kodu pośredniczącego. W języku PL/I zadeklaruj tylko nazwy wywołań używane w programie.

Tabela 159. Nazwy połączeń dla połączeń dynamicznych

Wywołanie MQI	Dynamiczne nazwy wywołań (innych niż RRS)	Nazwy dynamicznych wywołań programu CICS	Nazwy dynamicznych wywołań programu IMS
MQBACK	CSQBBACK	nieobsługiwane	Nieobsługiwane
MQBUFMH	CSQBBFMH	CSQCBFMH [1]	MQBUFMH
MQCB	CSQBCB	CSQCCB [1]	Nieobsługiwane
MQCLOSE	CSQBCLOS	CSQCCLOS	MQCLOSE
MQCMIT	CSQBCOMM	nieobsługiwane	Nieobsługiwane
MQCONN	CSQBCONN	CSQCCONN	MQCONN

Tabela 159. Nazwy połączeń dla połączeń dynamicznych (kontynuacja)			
Wywołanie MQI	**Dynamiczne nazwy wywołań (innych niż RRS)**	**Nazwy dynamicznych wywołań programu CICS**	**Nazwy dynamicznych wywołań programu IMS**
MQCONNX	CSQBCONX	CSQCCONX	MQCONNX
MQCRTMH	CSQBCTMH	CSQCCTMH [1]	MQCRTMH
MQCTL	CSQBCTL	CSQCCTL [1]	Nieobsługiwane
MQDISC	CSQBDISC	CSQCDISC	MQDISC
MQDLTMH	CSQBDTMH	CSQCDTMH [1]	MQDLTMH
MQDLTMP	CSQBDTMP	CSQCDTMP [1]	MQDLTMP
MQGet	CSQBGET	CSQCGET	MQGET
MQINQ	CSQBINQ	CSQCINQ	MQINQ
MQINQMP	CSQBIQMP	CSQCIQMP [1]	MQINQMP
MQMHBUF	CSQBMHBF	CSQCMHBF [1]	MQMHBUF
MQOPEN	CSQBOPEN	CSQCOPEN	MQOPEN
MQPUT	CSQBPUT	CSQCPUT	MQPUT
MQPUT1	CSQBPUT1	CSQCPUT1	MQPUT1
MQSET	CSQBSET	CSQCSET	MQSET
MQSETMP	CSQBSTMP	CSQCSTMP [1]	MQSETMP
MQSTAT	CSQBSTAT	CSQCSTAT [1]	MQSTAT
MQSUB	CSQBSUB	CSQCSUB [1]	MQSUB
MQSUBRQ	CSQBSUBR	CSQCSUBR [1]	MQSUBRQ

Uwaga: 1. Te wywołania funkcji API są dostępne tylko wtedy, gdy używany jest system 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 można znaleźć w następujących rysunkach:

- Wsadowy i COBOL: patrz Rysunek 122 na stronie 1138
- CICS i COBOL: patrz Rysunek 123 na stronie 1138
- IMS i COBOL: patrz Rysunek 124 na stronie 1138
- Wsadowy i asembler: patrz Rysunek 125 na stronie 1139
- CICS i asembler: patrz Rysunek 126 na stronie 1139
- IMS i asembler: patrz Rysunek 127 na stronie 1139
- Wsadowy i C: Rysunek 128 na stronie 1139
- CICS i C: patrz Rysunek 129 na stronie 1139
- IMS i C: patrz Rysunek 130 na stronie 1140
- Wsadowy i PL/I: patrz Rysunek 131 na stronie 1140
- IMS i PL/I: patrz Rysunek 132 na stronie 1140

```
...
       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 122. Dynamiczne łączenie za pomocą 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 123. Dynamiczne łączenie za pomocą 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 124. Dynamiczne łączenie za pomocą języka COBOL w środowisku IMS

```
...
         LOAD   EP=CSQBOPEN
...
         CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
         DELETE EP=CSQBOPEN
...
```

Rysunek 125. 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 126. Dynamiczne łączenie przy użyciu języka asemblacji w środowisku produktu CICS

```
...
         LOAD   EP=MQOPEN
...
         CALL (15),(HCONN,MQOD,OPTIONS,HOBJ,COMPCODE,REASON),VL
...
         DELETE EP=MQOPEN
...
```

Rysunek 127. Dynamiczne łączenie przy użyciu języka asemblacji w środowisku produktu 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 128. 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 129. Dynamiczne łączenie za pomocą 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 130. Dynamiczne łączenie za pomocą 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 131. Dynamiczne łączenie przy użyciu 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 132. Dynamiczne łączenie przy użyciu języka PL/I w środowisku IMS

z/OS *Uwagi dotyczące RRS*

Należy rozważyć użycie tych informacji, jeśli program użytkowy korzysta z usługi RRS w celu koordynowania aktualizacji.

Produkt IBM MQ udostępnia dwa różne kody pośredniczące dla programów wsadowych, które wymagają koordynacji RRS-patrz “Adapter zadania wsadowego RRS” na stronie 986. Różnica w zachowaniu późniejszych wywołań funkcji API jest określana na podstawie czasu MQCONN przez adapter zadania wsadowego z informacji przekazywanych przez procedurę pośredniczenia w interfejsie API MQCONN lub MQCONNX. Oznacza to, że dynamiczne wywołania interfejsu API są dostępne dla programów wsadowych, które wymagają koordynacji usługi RRS, pod warunkiem że początkowe połączenie z produktem IBM MQ zostało wykonane przy użyciu odpowiedniego kodu pośredniczącego. Poniższy przykład ilustruje to:

```
       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*

Ta sekcja zawiera informacje na temat debugowania programów TSO i CICS oraz informacji o śledzeniu śledzenia produktu CICS .

Główne pomoce do debugowania programów aplikacji IBM MQ for z/OS to kody przyczyny zwracane przez każde wywołanie funkcji API. Listę tych czynności, w tym pomysłów na działania naprawcze, można znaleźć w następujących celach:

- Komunikaty systemu IBM MQ for z/OS , kody zakończenia i kody przyczyny dla IBM MQ for z/OS
- Komunikaty i kody przyczyny dla wszystkich pozostałych platform IBM MQ

W tym temacie opisano również inne narzędzia do debugowania, które mogą być używane w określonych środowiskach.

Debugowanie programów TSO

Dla programów TSO dostępne są następujące interaktywne narzędzia do debugowania:

- TEST, narzędzie
- Interaktywne narzędzie do debugowania VS COBOL II
- Interaktywne narzędzie do debugowania INSPECT dla programów w języku C i PL/I

Debugowanie programów CICS

Program CICS Execution Diagnostic Facility (CEDF) służy do interaktywnego testowania programów CICS bez konieczności modyfikowania programu lub procedury przygotowawczego przygotowania programu.

Więcej informacji na temat EDF znajduje się w publikacji *CICS Transaction Server for z/OS CICS Application Programming Guide*.

CICS ślad

Prawdopodobnie pomocne będzie użycie transakcji CICS Śledzenie transakcji (CETR) do sterowania działaniem śledzenia CICS .

Więcej informacji na temat CETR zawiera podręcznik *CICS Transaction Server for z/OS CICS-Dostarczone transakcje* .

Aby określić, czy śledzenie CICS jest aktywne, wyświetl status połączenia przy użyciu panelu CKQC. Na tym panelu wyświetlany jest również numer śledzenia.

Informacje na temat interpretacji pozycji śledzenia produktu CICS zawiera sekcja Tabela 160 na stronie 1142.

Pozycja śledzenia CICS dla tych wartości to AP0 *xxx* (gdzie *xxx* to numer śledzenia określony podczas włączenia adaptera CICS). Wszystkie pozycje śledzenia z wyjątkiem CSQCTEST są wydawane przez CSQCTRUE. Komenda CSQCTEST jest wydawana przez CSQCRST i CSQCDSP.

Tabela 160. Pozycje śledzenia adaptera CICS

Nazwa	Opis	Sekwencja śledzenia	Dane śledzenia
CSQCABNT	Nieprawidłowe zakończenie	Przed wywołaniem END_THREAD NIEPRAWIDŁOWYM do IBM MQ. Dzieje się tak dlatego, że zakończenie zadania i niejawne wycofania mogą być wykonane przez aplikację. Żądanie ROLLBACK jest zawarte w wywołaniu END_THREAD w tym przypadku.	Informacje o jednostce pracy. Informacji tych można użyć podczas znajdowania informacji o statusie pracy. (Na przykład można sprawdzić, czy dane wyjściowe są generowane przez komendę DISPLAY THREAD, czy program narzędziowy do drukowania dziennika produktu IBM MQ for z/OS).
CSQCBACK	Wycofaj punkt synchronizacji	Przed wprowadzeniem komendy BACKOUT do produktu IBM MQ for z/OS. Wynika to z jawnego żądania 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 wysłaniem instrukcji COMMIT do IBM MQ for z/OS. Może to być spowodowane żądaniem zatwierdzenia jednofazowego lub drugą fazą żądania zatwierdzania dwufazowego. Żądanie wynika z jawnego żądania punktu synchronizacji z aplikacji.	Informacje o jednostce pracy.
CSQCEXER	Wykonaj rozstrzygnięcie	Przed wydaniem komendy EXECUTE_RESOLVE na serwerze IBM MQ for z/OS.	Informacje o jednostce pracy jednostki pracy wydające rozwiązanie EXECUTE_RESOLVE. Jest to ostatnia wątpliwa jednostka pracy w procesie resynchronizacji.
CSQCGETW	Oczekiwanie na GET	Przed uruchomieniem programu CICS należy poczekać.	Adres EBC, na który należy czekać.
CSQCGMGD	Pobierz dane komunikatu	Po pomyślnym powrocie z wywołania MQGET.	Maksymalnie 40 bajtów danych komunikatu.
CSQCGMGH	Uchwyt komunikatu GET	Przed wprowadzeniem komendy MQGET do produktu IBM MQ for z/OS.	Uchwyt obiektu.
CSQCGMGI	Pobierz identyfikator komunikatu	Po pomyślnym powrocie z wywołania MQGET.	Identyfikator komunikatu i identyfikator korelacji komunikatu.
CSQCINDL	Lista wątpliwych	Po pomyślnym powrocie z drugiej INQUIRE_INDOUBT.	Lista wątpliwych jednostek pracy.
CSQCINDO	Tylko IBM		

Tabela 160. Pozycje śledzenia adaptera CICS (kontynuacja)

Nazwa	Opis	Sekwencja śledzenia	Dane śledzenia
CSQCINDS	Wielkość listy wątpliwych	Po pomyślnym powrocie z pierwszej INQUIRE_INDOUBT i lista wątpliwa nie jest pusta.	Długość listy. Podzielone przez 64 daje liczbę wątpliwych jednostek pracy.
CSQCINQH	Uchwyt INQ	Przed wydaniem komendy MQINQ w produkcie IBM MQ for z/OS.	Uchwyt obiektu.
CSQCLOSH	Uchwyt CLOSE	Przed wydaniem komendy MQCLOSE na serwerze IBM MQ for z/OS.	Uchwyt obiektu.
CSQCLOST	Utrata rozporządzania	Podczas procesu resynchronizacji program CICS informuje adapter o tym, że został zrestartowany, więc nie są dostępne żadne informacje o rozporządzaniu dotyczące jednostki pracy, która jest resynchronizowana.	Identyfikator jednostki pracy znany CICS dla jednostki pracy, która jest resynchronizowana.
CSQCNIND	Dyspozycja nie jest wątpliwa	Podczas procesu resynchronizacji program CICS informuje adapter, że jednostka pracy, która jest resynchronizowana, nie powinna być wątpliwa (co oznacza, że być może jest nadal uruchomiona).	Identyfikator jednostki pracy znany CICS dla jednostki pracy, która jest resynchronizowana.
CSQCNORT	Normalne zakończenie	Przed wprowadzeniem END_THREAD NORMAL do IBM MQ for z/OS. Jest to spowodowane zakończeniem zadania, a zatem aplikacja może wykonać niejawne zatwierdzenie punktu synchronizacji. Żądanie COMMIT jest zawarte w wywołaniu END_THREAD w tym przypadku.	Informacje o jednostce pracy.
CSQCOPNH	Uchwyt OPEN	Po pomyślnym powrocie z komendy MQOPEN.	Uchwyt obiektu.
CSQCOPNO	OPEN, obiekt	Przed wydaniem komendy MQOPEN dla produktu IBM MQ for z/OS.	Nazwa obiektu.
CSQCPMGD	Dane komunikatu PUT	Przed wprowadzeniem komendy MQPUT do produktu IBM MQ for z/OS.	Maksymalnie 40 bajtów danych komunikatu.
CSQCPMGH	Uchwyt komunikatu PUT	Przed wprowadzeniem komendy MQPUT do produktu IBM MQ for z/OS.	Uchwyt obiektu.
CSQCPMGI	Identyfikator komunikatu PUT	Po pomyślnym zakończeniu operacji MQPUT z produktu IBM MQ for z/OS.	Identyfikator komunikatu i identyfikator korelacji komunikatu.

Tabela 160. Pozycje śledzenia adaptera CICS (kontynuacja)

Nazwa	Opis	Sekwencja śledzenia	Dane śledzenia
CSQCPREP	Przygotowanie punktu synchronizacji	Przed wprowadzeniem produktu PREPARE do wersji IBM MQ for z/OS w pierwszej fazie przetwarzania zatwierdzania dwufazowego. To wywołanie można również wywołać z rozproszonego komponentu kolejkowania w postaci wywołania funkcji API.	Informacje o jednostce pracy.
CSQCP1MD	Dane komunikatu PUTONE	Przed wydaniem komendy MQPUT1 do IBM MQ for z/OS.	Do 40 bajtów danych komunikatu.
CSQCP1MI	ID komunikatu PUTONE	Po pomyślnym powrocie z MQPUT1.	Identyfikator komunikatu i identyfikator korelacji komunikatu.
CSQCP1ON	Nazwa obiektu PUTONE	Przed wydaniem komendy MQPUT1 do IBM MQ for z/OS.	Nazwa obiektu.
CSQCRBAK	Rozstrzygnięte wycofania	Przed wydaniem komendy RESOLVE_ROLLBACK do wersji IBM MQ for z/OS.	Informacje o jednostce pracy.
CSQCRCMT	Rozstrzygnięte zatwierdzenie	Przed wprowadzeniem RESOLVE_COMMIT do IBM MQ for z/OS.	Informacje o jednostce pracy.
CSQCRMIR	Odpowiedź RMI	Przed zwróceniem do interfejsu CICS RMI (interfejs menedżera zasobów) z konkretnego wywołania.	Architected wartość odpowiedzi RMI. Jego znaczenie jest zależne od typu wywołania. Te wartości zostały opisane w publikacji *CICS Transaction Server for z/OS Customization Guide*. Aby określić typ wywołania, należy przejrzeć poprzednie wpisy śledzenia wygenerowane przez komponent RMI produktu CICS .
CSQCRSYN	Ponowna synchronizacja	Przed rozpoczęciem procesu resynchronizacji dla zadania.	Identyfikator jednostki pracy znany CICS dla jednostki pracy, która jest resynchronizowana.
CSQCSETH	Uchwyt SET	Przed wydaniem komendy MQSET w produkcie IBM MQ for z/OS.	Uchwyt obiektu.
CSQCTASE	Tylko IBM		
CSQCTEST	Test śledzenia	Używany w wywołaniu EXEC CICS ENTER TRACE w celu sprawdzenia numeru śledzenia dostarczonego przez użytkownika lub statusu śledzenia połączenia.	Brak danych.
CSQCDCFF	Tylko IBM		

Obsługa proceduralnych błędów programu

Te informacje wyjaśniają błędy związane z wywołaniami MQI aplikacji, gdy wywołuje połączenie, lub gdy jego komunikat jest dostarczany do miejsca docelowego.

Jeśli jest to możliwe, menedżer kolejek zwraca wszystkie błędy, gdy tylko zostanie wykonane wywołanie MQI. Są to *błędy określone lokalnie*.

Podczas wysyłania komunikatów do kolejki zdalnej błędy mogą nie być widoczne podczas wywołania MQI. W tym przypadku menedżer kolejek, który identyfikuje błędy, zgłasza je, wysyłając kolejny komunikat do programu inicjującego. Są to *zdalnie określone błędy*.

Błędy określane lokalnie

Informacje o lokalnie określonych błędach, które obejmują: niepowodzenie wywołania MQI, przerwania systemowe i komunikaty zawierające niepoprawne dane.

Trzy najczęstszych przyczyn błędów, które menedżer kolejek może zgłosić natychmiast, są następujące:

- Niepowodzenie wywołania MQI; na przykład, ponieważ kolejka jest pełna
- Przerwa w uruchamianiu pewnej części systemu, od której zależy aplikacja, na przykład menedżer kolejek
- Komunikaty zawierające dane, których nie można pomyślnie przetworzyć

Jeśli używany jest asynchroniczny obiekt put, błędy nie są zgłaszane natychmiast. Użyj wywołania MQSTAT, aby pobrać informacje o statusie poprzednich asynchronicznych operacji put.

Niepowodzenie wywołania MQI

Menedżer kolejek może natychmiast zgłosić jakiekolwiek błędy w kodzie wywołania MQI. Wykonuje to za pomocą zestawu predefiniowanych kodów powrotu. Są one podzielone na kody zakończenia i kody przyczyny.

Aby określić, czy wywołanie powiodło się, menedżer kolejek zwraca *kod zakończenia* po zakończeniu wywołania. Istnieją trzy kody zakończenia, wskazujące powodzenie, częściowe zakończenie i niepowodzenie wywołania. Menedżer kolejek zwraca także *kod przyczyny* , który wskazuje przyczynę częściowego zakończenia lub niepowodzenia wywołania.

Kody zakończenia i przyczyny dla każdego wywołania są wyświetlane wraz z opisem tego wywołania w sekcji Kody powrotu. Więcej szczegółowych informacji, w tym pomysłów na działania naprawcze, można znaleźć w:

- 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 pozostałych platform IBM MQ

Zaprojektuj programy tak, aby obsługiwały wszystkie kody powrotu, które mogą pojawić się w każdym wywołaniu.

System interruptions

Jeśli menedżer kolejek, z którym jest on połączony, musi odzyskać od awarii systemu, aplikacja użytkownika może nie mieć żadnych informacji o przerwaniu. Należy jednak zaprojektować aplikację, aby upewnić się, że dane nie zostaną utracone, jeśli wystąpi taka przerwa.

Metody, których można użyć w celu upewnia się, że dane pozostają spójne, zależy od platformy, na której działa menedżer kolejek:

z/OS 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 punkty synchronizacji muszą być deklarowane przy użyciu:

- Wywołania IBM MQ for z/OS MQCMIT i MQBACK (patrz “Zatwierdzanie i wycofywanie jednostek pracy” na stronie 946), lub
- Usługi z/OS Transaction Management i Recoverable Resource Manager Services (RRS) umożliwiają obsługę dwufazowego punktu synchronizacji. Usługa RRS umożliwia aktualizowanie zarówno IBM MQ , jak i innych zasobów produktu z obsługą RRS, takich jak zasoby procedury składowanej Db2 , w ramach jednej logicznej jednostki pracy. Informacje na temat obsługi punktów synchronizacji RRS można znaleźć w sekcji “Usługi zarządzania transakcjami i odtwarzalne usługi menedżera zasobów” na stronie 951.

IBM i

Wywołania MQPUT i MQGET można wykonywać w ramach globalnych jednostek pracy, które są zarządzane przez kontrolę transakcji produktu IBM i . Punkty synchronizacji można deklarować za pomocą rodzimych komend IBM i COMMIT i ROLLBACK lub komend specyficznych dla języka. Lokalne jednostki pracy są zarządzane przez produkt IBM MQ przy użyciu wywołań MQCMIT i MQBACK.

UNIX, Linux, and Windows systemy

W tych środowiskach można wykonywać wywołania MQPUT i MQGET w zwykły sposób, ale punkty synchronizacji muszą być deklarowane za pomocą wywołań MQCMIT i MQBACK (patrz sekcja “Zatwierdzanie i wycofywanie jednostek pracy” na stronie 946). W środowisku CICS komendy MQCMIT i MQBACK są wyłączone, ponieważ można tworzyć wywołania MQPUT i MQGET w ramach jednostek pracy zarządzanych przez produkt CICS.

Użyj trwałych komunikatów do przenoszenia wszystkich danych, których nie można sobie pozwolić na utratę danych. Komunikaty trwałe są przywracane w kolejkach, jeśli menedżer kolejek ma do odtworzenia po awarii. ULW W przypadku produktu IBM MQ w systemie UNIX, Linux, and Windowswywołanie MQGET lub MQPUT w obrębie aplikacji nie powiedzie się w momencie zapełnienia wszystkich plików dziennika, za pomocą komunikatu MQRC_RESOURCE_PROBLEM. Więcej informacji na temat plików dziennika w systemie UNIX, Linux, and Windowszawiera sekcja Administrowanie.

z/OS W przypadku produktu z/OS patrz Planowanie w systemie z/OS.

Jeśli menedżer kolejek jest zatrzymany przez operatora podczas działania aplikacji, zwykle używana jest opcja wygaszania. Menedżer kolejek przechodzi w stan wygaszania, w którym aplikacje mogą kontynuować pracę, ale muszą zakończyć się tak szybko, jak jest to wygodne. Małe, szybkie aplikacje mogą prawdopodobnie ignorować stan wygaszania i kontynuować aż do zakończenia pracy w normalnym trybie. Dłuższe działające aplikacje lub te, które oczekują na nadejście komunikatów, powinny używać opcji *fail if quiescing* (niepowodzenie w przypadku wygaszania), gdy są używane wywołania MQOPEN, MQPUT, MQPUT1i MQGET. Te opcje oznaczają, że wywołania nie powiodą się, gdy menedżer kolejek wygaśnie, ale aplikacja może nadal mieć czas na zakończenie czyszczenia, wydając wywołania, które ignorują stan wygaszania. Takie aplikacje mogą również zatwierdzić lub wycofać zmiany, które zostały wprowadzone, a następnie zakończyć.

Jeśli menedżer kolejek jest zmuszony do zatrzymania (czyli zatrzymania bez wygaszania), aplikacje odbierają kod przyczyny MQRC_CONNECTION_BROKEN podczas tworzenia wywołań MQI. Wyjdź z aplikacji lub, alternatywnie, w systemach IBM i IBM MQ for IBM i, UNIX, Linux, and Windows , wywołaj wywołanie MQDISC.

Komunikaty zawierające niepoprawne dane

Jeśli w aplikacji używane są jednostki pracy, jeśli program nie może pomyślnie przetworzyć komunikatu, który pobiera z kolejki, zostanie utworzona kopia zapasowa wywołania MQGET.

Menedżer kolejek przechowuje licznik (w polu *BackoutCount* deskryptora komunikatu) liczby przypadków, które się zdarzy. Ta liczba jest zachowana w deskryptorze każdego komunikatu, który ma wpływ na działanie. Licznik ten może dostarczyć cennych informacji na temat efektywności aplikacji. Komunikaty z licznymi wycofami, które zwiększają się w czasie, są wielokrotnie odrzucane; projektuj aplikację tak, aby analizował przyczyny tego działania i odpowiednio obsługuje takie komunikaty.

z/OS W systemie IBM MQ for z/OS, aby liczba wycofań została zachowana ponownie dla menedżera kolejek, należy ustawić atrybut **HardenGetBackout** na wartość MQQA_BACKOUT_HARDENED; w przeciwnym razie, jeśli menedżer kolejek musi zostać zrestartowany, nie zachowuje dokładnego licznika wycofań dla każdego komunikatu. Ustawienie atrybutu w ten sposób powoduje dodanie kary dodatkowego przetwarzania.

W systemie IBM MQ dla systemów IBM i IBM i, Windowsi UNIX and Linux liczba wycofań zawsze jest zachowana po restarcie menedżera kolejek.

z/OS Ponadto w systemie IBM MQ for z/OSpo usunięciu komunikatów z kolejki w ramach jednostki pracy można oznaczyć jeden komunikat, aby nie był on ponownie dostępny, jeśli jednostka pracy jest wycofana przez aplikację. Oznaczony komunikat jest traktowany tak, jakby został pobrany w ramach nowej jednostki pracy. Oznacza to, że komunikat ma pomijać wycofany dostęp przy użyciu opcji MQGMO_MARK_SKIP_BACKOUT.(w strukturze MQGMO), gdy używany jest wywołanie MQGET. Więcej informacji na temat tej techniki można znaleźć w sekcji “Pomijanie wycofania” na stronie 891 .

Korzystanie z komunikatów raportu w celu określenia problemu

Menedżer kolejek zdalnych nie może zgłaszać błędów, takich jak niepowodzenie umieszczenia komunikatu w kolejce podczas wywoływania interfejsu MQI, ale może wysłać komunikat raportu, aby stwierdzić, w jaki sposób został przetworzony komunikat.

W ramach aplikacji można tworzyć (MQPUT) komunikaty raportów, a także wybierać opcję ich odbierania (w takim przypadku są one wysyłane przez inną aplikację lub przez menedżera kolejek).

Tworzenie komunikatów raportu

Komunikaty raportów umożliwiają aplikacji poinformowanie innej aplikacji o tym, że nie może zajmować się wysłanym komunikatem.

Jednak początkowo pole *Report* musi być analizowane w celu określenia, czy aplikacja, która wysłała komunikat, jest zainteresowana informowaniu o problemach. Po ustaleniu, że wymagany jest komunikat raportu, należy podjąć decyzję:

- Określa, czy ma zostać dołączona cała oryginalna wiadomość, tylko pierwsze 100 bajtów danych, czy też żaden z pierwotnych komunikatów.
- Co zrobić z oryginalnym komunikatem. Można je odrzucić lub pozwolić na przejście do kolejki niedostarczonych komunikatów.
- Określa, czy zawartość pól *MsgId* i *CorrelId* jest również potrzebna.

Użyj pola *Feedback* , aby wskazać przyczynę wygenerowania komunikatu raportu. Umieszczanie komunikatów raportu w kolejce odpowiedzi aplikacji. Więcej informacji na ten temat zawiera sekcja Opinia .

Żądanie i odbieranie (MQGET) komunikatów raportów

Po wysłaniu wiadomości do innej aplikacji użytkownik nie jest informowany o żadnych problemach, chyba że zostanie zakończone pole *Report* w celu wskazania, że wymagana jest opinia. Informacje na temat dostępnych opcji zawiera sekcja Struktura pola raportu .

Menedżery kolejek zawsze umieszczają komunikaty raportów w kolejce odpowiedzi aplikacji, dlatego zalecane jest, aby własne aplikacje były takie same. Jeśli używany jest mechanizm komunikatów raportu, należy określić nazwę kolejki odpowiedzi w deskryptorze komunikatu komunikatu. W przeciwnym razie wywołanie MQPUT nie powiedzie się.

Aplikacja musi zawierać procedury, które monitorują kolejkę odpowiedzi i przetwarzali wszystkie komunikaty, które są na nim przesyłane. Należy pamiętać, że komunikat raportu może zawierać całą oryginalną wiadomość, pierwsze 100 bajtów oryginalnego komunikatu lub żaden z oryginalnych komunikatów.

Menedżer kolejek ustawia pole *Feedback* komunikatu raportu w celu wskazania przyczyny błędu. Na przykład kolejka docelowa nie istnieje. Programy powinny być takie same.

Więcej informacji na temat komunikatów raportów zawiera sekcja “Komunikaty raportów” na stronie 19.

Zdalnie określone błędy

Podczas wysyłania komunikatów do kolejki zdalnej, nawet jeśli lokalny menedżer kolejek przetworzy wywołanie MQI bez znalezienia błędu, inne czynniki mogą mieć wpływ na sposób obsługi komunikatu przez zdalny menedżer kolejek.

Na przykład kolejka kierowana przez użytkownika może być pełna lub nawet nie istnieje. Jeśli komunikat musi być obsługiwany przez inne pośrednie menedżery kolejek na trasie do kolejki docelowej, to każdy z tych menedżerów może znaleźć błąd.

Problemy z dostarczeniem komunikatu

Gdy wywołanie MQPUT nie powiedzie się, można spróbować ponownie umieścić komunikat w kolejce, zwrócić go do nadawcy lub umieścić w kolejce niedostarczonych komunikatów.

Każda opcja ma swoje zalety, ale może nie być konieczne ponowne umieszczanie komunikatu, jeśli przyczyną niepowodzenia operacji MQPUT było to, że kolejka docelowa była pełna. W tym przypadku umieszczenie go w kolejce niedostarczonych komunikatów pozwala na dostarczenie go do właściwej kolejki docelowej w późniejszym czasie.

Ponowna próba dostarczenia komunikatu

Zanim komunikat zostanie umieszczony w kolejce niedostarczonych komunikatów, zdalny menedżer kolejek próbuje ponownie umieścić komunikat w kolejce, jeśli atrybuty *MsgRetryCount* i *MsgRetryInterval* zostały ustawione dla kanału, lub jeśli istnieje program obsługi wyjścia ponawiania, który ma być używany (którego nazwa jest wstrzymana w polu *MsgRetryExitId* atrybutu kanału).

Jeśli pole *MsgRetryExitId* jest puste, używane są wartości z atrybutów *MsgRetryCount* i *MsgRetryInterval* .

Jeśli pole *MsgRetryExitId* nie jest puste, zostanie uruchomiony program obsługi wyjścia o tej nazwie. Więcej informacji na temat korzystania z własnych programów obsługi wyjścia zawiera sekcja “Programy obsługi wyjścia kanału dla kanałów przesyłania komunikatów” na stronie 1059.

Zwróć wiadomość do nadawcy

Użytkownik zwraca komunikat do nadawcy, żądając wygenerowania komunikatu raportu, który ma zawierać wszystkie oryginalne wiadomości.

Szczegółowe informacje na temat opcji komunikatów raportu zawiera sekcja “Komunikaty raportów” na stronie 19 .

Korzystanie z kolejki niedostarczonych komunikatów (niedostarczonych komunikatów)

Gdy menedżer kolejek nie może dostarczyć komunikatu, próbuje on umieścić komunikat w jego kolejce niedostarczonych komunikatów. Ta kolejka powinna być zdefiniowana podczas instalowania menedżera kolejek.

Programy mogą korzystać z kolejki niedostarczonych komunikatów w taki sam sposób, w jaki korzysta z niego menedżer kolejek. Nazwę kolejki niedostarczonych komunikatów można znaleźć, otwierając obiekt menedżera kolejek (przy użyciu wywołania MQOPEN) i zapytaj o atrybut **DeadLetterQName** (przy użyciu wywołania MQINQ).

Gdy menedżer kolejek umieszcza komunikat w tej kolejce, dodaje nagłówek do komunikatu, którego format jest opisany przez strukturę nagłówka niedostarczonych komunikatów (MQDLH). Patrz sekcja MQDLH-Nagłówek niedostarczonych komunikatów. Nagłówek ten zawiera nazwę kolejki docelowej oraz przyczynę umieszczenia komunikatu w kolejce niedostarczonych komunikatów. Musi on zostać usunięty, a problem musi zostać rozwiązany, zanim komunikat zostanie umieszczony w przeznaczonej dla niej

kolejce. Ponadto menedżer kolejek zmienia pole *Format* deskryptora komunikatu (MQMD) w celu wskazania, że komunikat zawiera strukturę MQDLH.

Struktura MQDLH

Zaleca się dodanie struktury MQDLH do wszystkich komunikatów umieszczonych w kolejce niedostarczonych komunikatów. Jeśli jednak planowane jest użycie procedury obsługi niedostarczonych komunikatów udostępnianej przez niektóre produkty IBM MQ , należy dodać strukturę MQDLH do komunikatów.

Dodanie nagłówka do komunikatu może spowodować zbyt długi komunikat dla kolejki niedostarczonych komunikatów, dlatego zawsze należy się upewnić, że komunikaty są krótsze od maksymalnej wielkości dozwolonej dla kolejki niedostarczonych komunikatów, przez co najmniej wartość stałej MQ_MSG_HEADER_LENGTH. Maksymalna wielkość komunikatów dozwolonych w kolejce jest określana na podstawie wartości atrybutu **MaxMsgLength** kolejki. W przypadku kolejki niedostarczonych komunikatów należy upewnić 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 komunikaty docierające do niego są przetwarzane. Procedura obsługi kolejki niedostarczonych komunikatów jest uruchamiana jako program narzędziowy wsadowy i może być używana do wykonywania różnych działań na wybranych komunikatach w kolejce niedostarczonych komunikatów. Więcej informacji na ten temat zawiera sekcja “Przetwarzanie kolejki niedostarczanych komunikatów” na stronie 1149.

Jeśli konieczna jest konwersja danych, menedżer kolejek przekształca informacje w nagłówku, gdy używana jest opcja MQGMO_CONVERT w wywołaniu MQGET. 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. Możliwe wskazanie tej sytuacji oznacza, że komunikaty w kolejce niedostarczonych komunikatów mają taką samą długość, jak wartość atrybutu **MaxMsgLength** kolejki.

Przetwarzanie kolejki niedostarczanych komunikatów
Informacje te zawierają ogólne informacje na temat interfejsu programistycznego podczas korzystania z przetwarzania w kolejce niedostarczonych komunikatów.

Przetwarzanie kolejki niedostarczanych komunikatów zależy od lokalnych wymagań systemowych, ale podczas rysowania specyfikacji należy rozważyć następujące kwestie:

- Komunikat może zostać zidentyfikowany jako posiadający nagłówek kolejki niedostarczonych komunikatów, ponieważ wartość pola formatu w deskryptyce MQMD to MQFMT_DEAD_LETTER_HEADER.
- W systemie IBM MQ for z/OS przy użyciu CICS, jeśli agent MCA umieszcza ten komunikat w kolejce niedostarczonych komunikatów, pole *PutApplType* ma wartość MQAT_CICS, a pole *PutApplName* jest *ApplId* systemu CICS , po którym następuje nazwa transakcji agenta MCA.
- Przyczyna, dla której komunikat ma być kierowany do kolejki niedostarczonych komunikatów, znajduje się w polu *Reason* nagłówka kolejki niedostarczonych komunikatów.
- Nagłówek kolejki niedostarczonych komunikatów zawiera szczegółowe informacje na temat nazwy kolejki docelowej i nazwy menedżera kolejek.
- Nagłówek kolejki niedostarczonych komunikatów zawiera pola, które muszą zostać przywrócone do deskryptora komunikatu, zanim komunikat zostanie umieszczony w kolejce docelowej. Są to:
  1. *Encoding*
  2. *CodedCharSetId*
  3. *Format*
- Deskryptor komunikatu jest taki sam jak deskryptor PUT przez oryginalną aplikację, z wyjątkiem trzech wyświetlonych pól (Encoding, CodedCharSetIdi Format).

Aplikacja kolejki niedostarczonych komunikatów musi wykonać jedną lub więcej z następujących czynności:

- Sprawdź pole *Reason* . Agent MCA może umieścić komunikat 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

    Przyczyna to dowolny kod przyczyny MQRC_ *, który może zostać zwrócony przez operację MQPUT .
  - Wyjście użytkownika zażądało tego działania

    Kodem przyczyny jest kod dostarczony przez program użytkownika lub domyślna wartość MQRC_SUPPRESSED_BY_EXIT
- Spróbuj przekazać komunikat do zamierzonego miejsca docelowego, gdzie jest to możliwe.
- Należy zachować wiadomość przez pewien czas przed usunięciem, gdy przyczyna dywersji jest określona, ale nie jest ona natychmiast korygowana.
- Podaj instrukcje dla administratorów, którzy rozwiążają problemy, jeśli zostały one określone.
- Odrzuć komunikaty, które są uszkodzone lub w inny sposób nieprzetwarzalne.

Istnieją dwa sposoby radzenia sobie z komunikatami, które zostały odzyskane z kolejki niedostarczonych komunikatów:

1. Jeśli komunikat dotyczy kolejki lokalnej:
   - Przeprowadzanie tłumaczeń kodu wymaganych do wyodrębnienia danych aplikacji
   - Przeprowadzanie konwersji kodu na tych danych, jeśli jest to funkcja lokalna
   - Umieść komunikat wynikowy w kolejce lokalnej ze wszystkimi szczegółami odtwarzanego deskryptora komunikatu
2. Jeśli komunikat jest przeznaczony dla kolejki zdalnej, należy umieścić komunikat w kolejce.

Informacje na temat sposobu obsługi niedostarczanych komunikatów w rozproszonym środowisku kolejkowania można znaleźć w sekcji Co się dzieje, gdy nie można dostarczyć komunikatu?.

Programowanie grupowe

Informacje zawarte w tej sekcji umożliwiają zapoznanie się z zadaniami programistycznym IBM MQ Multicast, takimi jak łączenie się z menedżerem kolejek i raportowanie wyjątków.

IBM MQ Rozsyłanie grupowe zostało zaprojektowane tak, aby było możliwie jak najbardziej przezroczyste dla użytkownika, a mimo to jest zgodne z istniejącymi aplikacjami. Zdefiniowanie obiektu COMMINFO i ustawienie parametrów **MCAST** i **COMMINFO** obiektu TOPIC oznacza, że istniejące aplikacje produktu IBM MQ nie wymagają istotnego przebudowowania w celu użycia rozsyłania grupowego. Jednak mogą wystąpić pewne ograniczenia (więcej informacji na ten temat zawiera sekcja “Rozsyłanie grupowe i MQI” na stronie 1150), a niektóre zagadnienia związane z bezpieczeństwem (więcej informacji na ten temat można znaleźć w temacie Multicast security).

Rozsyłanie grupowe i MQI

Te informacje umożliwiają zrozumienie głównych pojęć związanych z interfejsem kolejki komunikatów (Message Queue Interface-MQI) oraz sposobu ich powiązania z programem IBM MQ Multicast.

Subskrypcje rozsyłania grupowego są nietrwałe, ponieważ nie ma żadnych aktywnych kolejek fizycznych, nie ma miejsca do przechowywania wiadomości w trybie bez połączenia, które są tworzone przez subskrypcje trwałe.

Po zasubskrybowaniu przez aplikację tematu rozsyłania grupowego nadawany jest uchwyt obiektu, z którego może korzystać lub MQGET, tak jakby był to uchwyt do kolejki. Oznacza to, że obsługiwane są tylko zarządzane subskrypcje rozsyłania grupowego (subskrypcje utworzone za pomocą komendy MQSO_MANAGED), tj. nie jest możliwe utworzenie subskrypcji i 'wskaż' komunikaty w kolejce. Oznacza

to, że komunikaty muszą być pobierane z uchwytu obiektu zwróconego w wywołaniu subskrypcji. Na kliencie komunikaty są zapisywane w buforze komunikatów do czasu ich konsumowania przez klienta. Więcej informacji na ten temat zawiera sekcja MessageBuffer sekcji pliku konfiguracyjnego klienta . Jeśli klient nie nadąże za pomocą szybkości publikowania, komunikaty są usuwane zgodnie z wymaganiami, a najstarsze komunikaty są usuwane jako pierwsze.

Zwykle jest to decyzja administracyjna, niezależnie od tego, czy aplikacja używa rozsyłania grupowego, czy nie, określonych przez ustawienie atrybutu MCAST dla obiektu TOPIC. Jeśli aplikacja publikowania musi upewnić się, że rozsyłanie grupowe nie jest używane, może użyć opcji `MQOO_NO_MULTICAST` . Podobnie aplikacja subskrybująca może zapewnić, że rozsyłanie grupowe nie jest używane przez zasubskrybowanie opcji `MQSO_NO_MULTICAST` .

IBM MQ Multicast obsługuje korzystanie z selektorów komunikatów. Selektor jest używany przez aplikację w celu zarejestrowania jej zainteresowania tylko tymi komunikatami o właściwościach, które spełniają zapytanie SQL92 , które reprezentuje łańcuch wyboru. Więcej informacji na temat selektorów komunikatów zawiera sekcja “Selektory” na stronie 30.

W poniższej tabeli przedstawiono wszystkie główne pojęcia MQI oraz sposób ich powiązania z Multicast:

Tabela 161. Pojęcia MQI i sposób ich powiązania z rozsyłaniem grupowym

Pojęcie MQI	**Działanie podczas próby przy użyciu 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	Akceptowany dla zarządzanego uchwytu. Tylko `Bieżące zapełnienie` może zostać zapytane.	• Jeśli wartością jest Bieżące zapełnienie, nie ma żadnego odpowiedniego kodu przyczyny. • Jeśli wartość jest inna niż Bieżące zapełnienie, kodem przyczyny jest 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
Przeglądanie komunikatów	Odrzucone	2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE
Blokowanie komunikatów	Odrzucone	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR

Tabela 161. Pojęcia MQI i sposób ich powiązania z rozsyłaniem grupowym (kontynuacja)

Pojęcie MQI	**Działanie podczas próby przy użyciu rozsyłania grupowego**	**Kod przyczyny**
Przeglądaj z zaznaczonym znakiem	Odrzucone	2036 (07F4) (RC2036): MQRC_NOT_OPEN_FOR_BROWSE
Przekaż kontekst	Odrzucone	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
MQPUT1	Odrzucono. Próba wykonania komendy MQPUT1 jest niepoprawna tylko w przypadku tematu rozsyłania grupowego (Multicast).	2560 (0A00) (RC2560): MQRC_MULTICAST_ONLY
subskrypcja trwała	Odrzucono, jeśli temat jest oznaczony jako "Tylko do rozsyłania grupowego", w przeciwnym razie jest dokonywana subskrypcja inna niż Multicast.	2436 (0984) (RC2436): MQRC_DURABILITY_NOT_ALLOWED
TopicString > 255	Odrzucono. Jeśli łańcuch tematu jest większy niż 255 znaków, zostanie on odrzucony w kliencie.	2425 (0979) (RC2425): MQRC_TOPIC_STRING_ERROR
Subskrypcja niezarządzana została wykonana	Odrzucono, jeśli temat jest oznaczony jako "Tylko do rozsyłania grupowego", w przeciwnym razie jest dokonywana subskrypcja inna niż Multicast.	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR
MQPMO_NOT_OWN_SUBS	Odrzucone	2046 (07FE) (RC2046): MQRC_OPTIONS_ERROR

Następujące elementy rozwijają się w niektórych pojęciach MQI z poprzedniej tabeli i zawierają informacje na temat niektórych pojęć MQI, które nie znajdują się w tabeli:

Trwałość komunikatu
W przypadku nietrwałych subskrybentów rozsyłania grupowego komunikaty trwałe pochodzące od publikatora są dostarczane w sposób nienaprawialny.

Obcinanie komunikatów
Obcinanie komunikatów jest obsługiwane, co oznacza, że aplikacja może wykonać następujące czynności:

1. Wprowadź komendę MQGET.
2. Pobieranie komendy MQRC_TRUNCATED_MSG_FAILED.
3. Przydziel większy bufor.
4. Ponownie wydaj 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 operacji IBM MQ Multicast continuous peer-to-peer; chociaż program IBM MQ łączy się z menedżerem kolejek produktu IBM MQ , komunikaty nie są przepływane 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 przepłyną przez menedżer kolejek. Oznacza to, że po zakończeniu operacji MQOPEN lub MQSUB w obiekcie tematu rozsyłania grupowego można kontynuować przesyłanie komunikatów rozsyłania grupowego, nawet jeśli połączenie z menedżerem kolejek zostało utracone. Istnieją dwa tryby pracy:

Do menedżera kolejek wykonywane jest normalne połączenie.
Komunikacja między rozsyłaniem grupowym jest możliwa, gdy istnieje połączenie z menedżerem kolejek. Jeśli nawiązanie połączenia nie powiedzie się, zostaną zastosowane normalne reguły MQI, na przykład: operacja MQPUT dla uchwytu obiektu rozsyłania grupowego zwraca wartość 2009 (07D9) (RC2009): MQRC_CONNECTION_BROKEN.

Nawiąże połączenie z menedżerem kolejek podczas ponownego nawiązywania połączenia klienta.
Komunikacja rozsyłania grupowego jest możliwa nawet podczas cyklu ponownego połączenia. Oznacza to, że nawet jeśli połączenie z menedżerem kolejek zostało zerwane, nie ma to wpływu na umieszczanie i korzystanie z komunikatów rozsyłania grupowego. Klient podejmuje próbę ponownego nawiązania połączenia z menedżerem kolejek. Jeśli ponowne nawiązanie połączenia nie powiedzie się, uchwyt połączenia zostanie zerwany, a wszystkie wywołania MQI, w tym rozgłaszanie, nie powiodą się. Więcej informacji na ten temat zawiera sekcja Automatyczne ponowne połączenie klienta .

Jeśli dowolna aplikacja jawnie zgłasza komendę MQDISC, wszystkie subskrypcje rozsyłania grupowego i uchwyty obiektów są zamykane.

Ciągłe grupowe operacje typu peer-to-peer

Jedną z zalet komunikacji równorzędnej między klientami jest to, że komunikaty nie muszą przepływać przez menedżer kolejek. W związku z tym, jeśli połączenie z menedżerem kolejek zostanie zerowane, przesyłanie komunikatów będzie kontynuowane. W przypadku wymagań dotyczących komunikatów ciągłych w tym trybie obowiązują następujące ograniczenia:

- Połączenie musi być nawiązane za pomocą jednej z opcji MQCNO_RECONNECT_ * w celu ciągłej pracy. Ten proces oznacza, że chociaż sesja komunikacyjna może być uszkodzona, rzeczywisty uchwyt połączenia nie jest zerwany i znajduje się w stanie ponownego połączenia. Jeśli ponowne nawiązanie połączenia nie powiedzie się, uchwyt połączenia zostanie zerwany, co uniemożliwia wykonanie wszystkich kolejnych wywołań MQI.

- W tym trybie obsługiwane są tylko zmaterializowane tabele MQPUT, MQGET, MQINQ i Async. Wszystkie czasowniki MQOPEN, MQCLOSE lub MQDISC wymagają ponownego nawiązania połączenia z menedżerem kolejek w celu zakończenia.
- Status przepływa do menedżera kolejek; w związku z tym każdy stan w menedżerze kolejek może być nieaktualny lub nie. Oznacza to, że klienty mogą wysyłać i odbierać komunikaty, a w menedżerze kolejek nie ma statusu znanego. Więcej informacji na ten temat zawiera sekcja Monitorowanie aplikacji Multicast

Konwersja danych w przesyłaniu komunikatów MQI na potrzeby rozsyłania grupowego

Informacje zawarte w tej sekcji umożliwiają zrozumienie sposobu działania konwersji danych w programie IBM MQ Multicast Messaging.

IBM MQ Multicast to współużytkowany, bezpołączeniowy protokół, a więc nie jest możliwe, aby każdy klient składał konkretne żądania konwersji danych. Każdy klient zasubskrybowany do tego samego strumienia rozsyłania grupowego odbiera te same dane binarne. Dlatego też, jeśli wymagana jest konwersja danych IBM MQ , konwersja jest wykonywana lokalnie na każdym kliencie.

Dane są przekształcane na kliencie w ruch sieciowy IBM MQ Multicast. Jeśli zostanie podana opcja **MQGMO_CONVERT** , konwersja danych jest wykonywana zgodnie z żądaniem. Formaty zdefiniowane przez użytkownika wymagają wyjścia konwersji danych zainstalowanego na kliencie. Informacje o bibliotekach znajdują się teraz w pakietach klienta i serwera. Informacje o bibliotekach znajdują się teraz w “Pisanie wyjść konwersji danych” na stronie 1082 .

Więcej informacji 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 w określeniu problemu przez wywołanie procedury obsługi zdarzeń w celu zgłaszania zdarzeń rozsyłania grupowego, które są raportowane za pomocą standardowego mechanizmu procedury obsługi zdarzeń IBM MQ .

Pojedyncze zdarzenie rozsyłania grupowego może spowodować wywołanie więcej niż jednego zdarzenia IBM MQ , ponieważ może istnieć wiele uchwytów połączeń produktu `MQHCONN` korzystających z tego samego nadajnika rozsyłania grupowego lub odbiornika. Jednak każdy wyjątek rozsyłania grupowego powoduje wywołanie tylko jednej procedury obsługi zdarzeń dla połączenia IBM MQ .

Stała IBM MQ `MQCBDO_EVENT_CALL` umożliwia aplikacjom zarejestrowanie wywołania zwrotnego w celu odbierania tylko zdarzeń produktu IBM MQ , a program `MQCBDO_MC_EVENT_CALL` umożliwia aplikacjom zarejestrowanie wywołania zwrotnego w celu odbierania tylko zdarzeń rozsyłania grupowego. Jeśli używane są obie stałe, odbierane są oba typy zdarzeń.

Żądanie zdarzeń rozsyłania grupowego

IBM MQ Zdarzenia rozsyłania grupowego używają stałej `MQCBDO_MC_EVENT_CALL` w polu `cbd.Options` . Poniższy przykład demonstruje 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 zostanie podana opcja MQCBDO_MC_EVENT_CALL , procedura obsługi zdarzeń jest wysyłana tylko do zdarzeń IBM MQ Multicast zamiast zdarzeń na poziomie połączenia. Aby zażądać, aby oba typy zdarzeń zostały wysłane 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 w polu Options zawiera sekcja Opcje (MQLONG).

Format zdarzenia rozsyłania grupowego

IBM MQ Wyjątki rozsyłania grupowego obejmują niektóre informacje dodatkowe 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 na krótki tekst opisu zdarzenia. W zależności od typu zdarzenia może być podana większa liczba parametrów. W poniższej tabeli przedstawiono wyjątki:

Tabela 162. Opisy kodów zdarzeń rozsyłania grupowego

Kod zdarzenia	Opis	Dane dodatkowe
MQMCEV_PACKET_LOSS	Nienaprawialna utrata pakietów	Liczba utraconych pakietów
MQMCEV_HEARTBEAT_TIMEOUT	Długa nieobecność pakietu kontrolnego pulsu	N/A
MQMCEV_VERSION_CONFLICT	Odbiór nowszych pakietów wersji protokołu	N/A
MQMCEV_RELIABILITY	Różne tryby niezawodności nadajnika i odbiornika	N/A
MQMCEV_CLOSED_TRANS	Transmisja tematu jest zamknięta przez 1 źródło	N/A
Błąd MQMCEV_STREAM_ERROR	Wykryto błąd w strumieniu	N/A
MQMCEV_NEW_SOURCE	Nowe źródło rozpoczyna transmisję w temacie	Struktura źródłowa
MQMCEV_RECEIVE_QUEUE_TRIMMED	Pakiety usunięte z pakietu PacketQ z powodu utraty ważności czasu lub miejsca	Liczba pakietów trimmed
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 **max_ack_retries** została przekroczona	Liczba usuniętych pakietów

Tabela 162. Opisy kodów zdarzeń rozsyłania grupowego (kontynuacja)

Kod zdarzenia	Opis	Dane dodatkowe
MQMCEV_STREAM_SUSPEND_NACK	NACKs zostało zawieszone w strumieniu zaakceptowany przez ten temat	Identyfikator strumienia zawieszenia Czas (w milisekundach), przez który strumień jest zawieszony
MQMCEV_STREAM_RESUME_NACK	NACKs zostały wznowione po zawieszeniu ich w strumieniu	Identyfikator strumienia
MQMCEV_STREAM_EXPELLED	Strumień zaakceptowany przez ten wątek został odrzucony ze względu na żądanie expel	Identyfikator strumienia
Komunikat MQMCEV_FIRST_MESSAGE	Pierwsza wiadomość od źródła	Numer komunikatu
Błąd MQMCEV_LATE_JOIN_FAILURE	Nie powiodło się uruchomienie opóźnionej sesji łączenia	N/A
MQMCEV_MESSAGE_LOSS	Nienaprawialna utrata komunikatów	Liczba utraconych komunikatów
Błąd MQMCEV_SEND_PACKET_FAILURE	Nie powiodło się wysłanie przez nadajnik rozsyłania grupowego pakietu rozsyłania grupowego	N/A
MQMCEV_REPAIR_DELAY	Odbiornik rozsyłania grupowego nie otrzymał pakietu naprawczego dla zaległego NAK	N/A
MQMCEV_MEMORY_ALERT_ON	Bufory odbiorcze odbiorcze zapełnia	Procent wykorzystania puli buforów
MQMCEV_MEMORY_ALERT_OFF	Bufory odbiorcze odbiorcze są wyłączone	Procent wykorzystania puli buforów
MQMCEV_NACK_ALERT_ON	Współczynnik żądań pakietu naprawczego odbiornika osiągnął wysoki wskaźnik poziomu	Bieżąca szybkość żądań napraw w pakietach na sekundę
MQMCEV_NACK_ALERT_OFF	Częstotliwość żądań pakietów napraw odbiornika jest wyłączona	Bieżąca szybkość żądań napraw w pakietach na sekundę
MQMCEV_REPAIR_ALERT_ON	Szybkość wysyłania pakietów przez nadajnik osiągnęła wysoki wskaźnik poziomu wody	N/A
MQMCEV_REPAIR_ALERT_OFF	Szybkość wysyłania pakietów do naprawy nadajnika jest wyłączona do normy	N/A
MQMCEV_SHM_DEST_UNUSABLE	Wykryto, że obszar pamięci współużytkowanej używany przez miejsce docelowe tematu nadajnika nie może być używany.	N/A

Tabela 162. Opisy kodów zdarzeń rozsyłania grupowego (kontynuacja)		
Kod zdarzenia	**Opis**	**Dane dodatkowe**
MQMCEV_SHM_PORT_UNUSABLE	Wykryto, że port pamięci współużytkowanej używany przez instancję odbiornika nie może być używany	N/A
Funkcja MQMCEV_CCT_GETTIME_FAILED	Nie powiodła się próba pobrania czasu z koordynowanego klastra	N/A
MQMCEV_DEST_INTERFACE_FAILURE	Interfejs sieciowy używany przez miejsce docelowe tematu nadajnika nie powiódł się, a zapasowy interfejs sieciowy jest niedostępny	
MQMCEV_DEST_INTERFACE_FAILOVER	Interfejs sieciowy używany przez miejsce docelowe tematu nadajnika nie powiódł się, a przełączenie awaryjne na inny interfejs zostało zakończone	
MQMCEV_PORT_INTERFACE-NIEPOWODZENIE	Interfejs sieciowy używany przez odbiornik rmmPort nie powiódł się, a zapasowy interfejs sieciowy jest niedostępny (lub też nie powiódł się)	KonfiguracjaRMM
MQMCEV_PORT_INTERFACE_FAILOVER	Interfejs sieciowy używany przez odbiornik rmmPort uległ awarii, a przełączenie awaryjne na inny interfejs zostało zakończone.	KonfiguracjaRMM

Kodowanie w języku C

Podczas kodowania programów IBM MQ w języku C. należy zwrócić uwagę na informacje zawarte w poniższych sekcjach.

Parametry wywołań MQI

Parametry, które są *tylko wejściowe* i typu MQHCONN, MQHOBJ, MQHMSG lub MQLONG, są przekazywane przez wartość. Dla wszystkich pozostałych parametrów *adres* parametru jest przekazywany przez wartość.

Nie wszystkie parametry, które są przekazywane przez adres, muszą być określone za każdym razem, gdy wywoływana jest funkcja. Jeśli określony parametr nie jest wymagany, jako parametr w wywołaniu funkcji można określić wskaźnik pusty, w miejsce adresu danych parametrów. Parametry, dla których jest to możliwe, są identyfikowane w opisach wywołań.

Żaden parametr nie jest zwracany jako wartość funkcji; w terminologii C oznacza to, że wszystkie funkcje zwracają wartość void.

Atrybuty tej funkcji są definiowane przez zmienną makra MQENTRY. Wartość tej zmiennej makra zależy od środowiska.

Parametry z niezdefiniowanym typem danych

Dla funkcji MQGET, MQPUT i MQPUT1 każdy z nich ma parametr **Buffer** , który ma niezdefiniowany typ danych. Ten parametr jest używany do wysyłania i odbierania danych komunikatu aplikacji.

Parametry tego sortowania są przedstawione w przykładach C jako tablice MQBYTE. Parametry można deklarować w ten sposób, ale zwykle bardziej wygodne jest zadeklarowanie ich jako struktury opisowej układu danych w komunikacie. Parametr funkcji jest zadeklarowany jako wskaźnik-do-void, a więc adres dowolnych 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 to nazwa podstawowego lub strukturalnego typu danych poprzedzona literą P w celu oznaczenia wskaźnika. Atrybuty wskaźnika są definiowane za pomocą zmiennej 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 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 te funkcje nie działają poprawnie z danymi zadeklarowani jako MQBYTE24.

Manipulowanie łańcuchami znaków

Gdy menedżer kolejek zwraca dane znakowe do aplikacji, menedżer kolejek zawsze dopełnia dane znakowe spacjami do zdefiniowanej długości pola. Menedżer kolejek nie zwraca łańcuchów zakończonych znakiem o kodzie zero, ale można użyć ich w danych wejściowych. Dlatego przy kopiowaniu, porównywaniu lub konkatenowaniu takich łańcuchów należy użyć funkcji łańcuchowych strncpy, strncmp lub strncat.

Nie należy używać funkcji łańcuchowych, które wymagają, aby łańcuch został zakończony znakiem o wartości NULL (strcpy, strcmp i strcat). Ponadto nie należy używać funkcji strlen w celu określenia długości łańcucha. Zamiast tego należy użyć funkcji sizeof, aby określić długość pola.

Wartości początkowe dla struktur

Plik włączany <cmqc.h> definiuje różne zmienne makra, których można użyć w celu udostępnienia początkowych wartości 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 MQI definiuje konkretne wartości, które są poprawne (na przykład w przypadku pól *StrucId* lub w polu *Format* w strukturze MQMD). Dla każdej z poprawnych wartości dostępne są dwie zmienne makra:

- Jedna zmienna makra definiuje wartość jako łańcuch o długości, z wyłączeniem niejawnej wartości NULL, która jest dokładnie zgodna z zdefiniowaną długością pola. Na przykład symbol ¬ reprezentuje pusty znak:

  ```
  #define MQMD_STRUC_ID "MD¬¬"
  #define MQFMT_STRING "MQSTR¬¬¬"
  ```

  Użyj tego formularza z funkcjami memcpy i memcmp.
- Inna zmienna makra definiuje wartość jako tablicę typu char; nazwa tej zmiennej makra to nazwa formularza łańcucha w postaci przyrostka _ARRAY. Na przykład:

  ```
  #define MQMD_STRUC_ID_ARRAY 'M','D','¬','¬'
  #define MQFMT_STRING_ARRAY 'M','Q','S','T','R','¬','¬','¬'
  ```

  Użyj tego formularza, aby zainicjować pole, gdy instancja struktury jest zadeklarowana z wartościami innymi niż te, które są udostępniane przez zmienną makra MQMD_DEFAULT.

Wartości początkowe dla struktur dynamicznych

Jeśli wymagana jest zmienna liczba instancji struktury, instancje są zwykle tworzone w głównej pamięci masowej uzyskanej dynamicznie przy użyciu funkcji calloc lub malloc.

Aby zainicjować pola w takich strukturach, zaleca się następujące techniki:

1. Zadeklaruj instancję struktury, używając odpowiedniej zmiennej makra MQxxx_DEFAULT w celu zainicjowania struktury. Ta instancja staje się *modelem* dla innych instancji:

   ```
   MQMD ModelMsgDesc = {MQMD_DEFAULT};
                                      /* declare model instance */
   ```

   Kod statyczny lub automatyczny słów kluczowych w deklaracji, aby nadać modelowi instancję statyczną lub dynamiczną, zgodnie z wymaganiami.
2. Użyj funkcji calloc lub malloc, aby uzyskać pamięć masową 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 języka C++

W przypadku języka programowania C++ pliki nagłówkowe zawierają następujące dodatkowe instrukcje, które są uwzględniane tylko wtedy, gdy używany jest kompilator C++:

```
#ifdef __cplusplus
  extern "C" {
#endif

/* rest of header file */

#ifdef __cplusplus
  }
#endif
```

Windows Kodowanie w Visual Basic

Informacje, które należy wziąć pod uwagę podczas kodowania programów IBM MQ w programie Microsoft Visual Basic. Produkt Visual Basic jest obsługiwany tylko w systemie Windows.

Uwaga: Z poziomu produktu 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 programu IBM WebSphere MQ 7.0 lub nowszego nie jest dostępna dla aplikacji VB. Jeśli programowanie jest prowadzone w wersji VB.NET, należy użyć klas IBM MQ dla produktu .NET. Więcej informacji na ten temat zawiera artykuł Tworzenie aplikacji .NET.

W produkcie IBM MQ 9.0obsługa produktu IBM MQ dla produktu Microsoft Visual Basic 6.0 jest nieaktualna. Zalecaną technologią zastępczą są klasy IBM MQ dla .NET.

Aby uniknąć niezamierzonego tłumaczenia danych binarnych przechodzących między Visual Basic i IBM MQ, należy użyć definicji MQBYTE zamiast MQSTRING. CMQB.BAS definiuje kilka nowych typów MQBYTE, które są odpowiednikami definicji bajtu w języku C i używają tych typów w strukturach produktu IBM MQ . Na przykład dla struktury MQMD (deskryptor komunikatu) wartość MsgId (identyfikator komunikatu) jest zdefiniowana jako MQBYTE24.

Produkt Visual Basic nie ma typu danych wskaźnika, dlatego odwołania do innych struktur danych produktu IBM MQ są przesunięte, a nie wskaźnikowe. Zadeklaruj strukturę złożoną składającą się z dwóch struktur komponentów i określ strukturę złożoną w wywołaniu. Obsługa produktu IBM MQ dla produktu Visual Basic udostępnia wywołanie MQCONNXAny w celu umożliwienia i umożliwienia aplikacjom klienckim określenia właściwości kanału w połączeniu klienta. Akceptuje on strukturę o nietypowym typie (MQCNOCD) w miejsce typowej struktury MQCNO.

Struktura MQCNOCD to struktura złożona składająca się z obiektu MQCNO, po którym następuje zmaterializowana tabela zapytania (MQCD). Ta struktura jest zadeklarowana w pliku nagłówkowych wyjść CMQXB. Użyj podprogramu MQCNOCD_DEFAULTS, aby zainicjować strukturę MQCNOCD. Udostępniono przykładowy proces tworzenia wywołań MQCONNX (amqscnxb.vbp).

Parametr MQCONNXAny ma te same parametry co parametr MQCONNX, z tą różnicą, że parametr **ConnectOpts** jest zadeklarowany jako typ dowolnego typu danych, a nie typ danych MQCNO. Umożliwia to akceptowanie przez funkcję struktury MQCNO lub struktury MQCNOCD. Ta funkcja jest zadeklarowana w głównym pliku nagłówkowy CMQB.

Pojęcia pokrewne

“Przygotowywanie programów Visual Basic w programie Windows” na stronie 1126
Informacje do rozważenia podczas korzystania z programów Microsoft Visual Basic w systemie Windows.

Odsyłacze pokrewne

“Łączenie aplikacji produktu Visual Basic z kodem produktu IBM MQ MQI client” na stronie 1016

Aplikacje produktu Microsoft Visual Basic można połączyć z kodem IBM MQ MQI client w systemie Windows.

Kodowanie w języku COBOL

Należy zwrócić uwagę na informacje zawarte w poniższej sekcji podczas kodowania programów IBM MQ w języku COBOL.

Stałe nazwane

Nazwy stałych są wyświetlane jako część nazwy, która zawiera znak podkreślenia (_). W języku COBOL należy użyć znaku łącznika (-) w miejsce podkreślenia. Stałe, które mają wartości łańcuchowe, używają jednego znaku 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, należy zadeklarować jawnie element level-01 , a następnie użyć instrukcji COPY do skopiowania w deklaracjach stałych:

```
WORKING-STORAGE SECTION.
01 MQM-CONSTANTS.
COPY CMQV.
```

Jednak ta metoda powoduje, że stałe zajmą pamięć masową w programie, nawet jeśli nie są one przywołane. Jeśli stałe są zawarte w wielu oddzielnych programach w obrębie tej samej jednostki wykonywania, wówczas będzie istnieć wiele kopii stałych; może to spowodować użycie znacznej ilości pamięci głównej. Aby tego uniknąć, należy dodać klauzulę GLOBAL do deklaracji level-01 :

```
* Declare a global structure to hold the constants
01 MQM-CONSTANTS GLOBAL.
COPY CMQV.
```

To przydziela pamięć tylko dla *jednego* zestawu stałych w jednostce uruchamiania. Stałe mogą jednak być przywoływani przez *dowolny* program w obrębie jednostki uruchamiania, a nie tylko program, który zawiera deklarację level-01 .

Zapewnianie wyrównania struktury

Należy zwrócić uwagę na zapewnienie, że struktury IBM MQ , które są przekazywane do uruchamiania w wywołaniu programu MQ , muszą być wyrównane do granic słów. Granica słowa to 4 bajty dla procesów 32-bitowych, 8 bajtów dla procesów 64-bitowych i 16 bajtów dla procesów 128-bitowych (IBM i).

Jeśli to możliwe, wszystkie struktury produktu IBM MQ należy umieścić razem, tak aby były wyrównane do granicy.

Kodowanie w języku asemblera System/390 (interfejs kolejki komunikatów)

Podczas kodowania programów IBM MQ for z/OS w języku asemblera należy zwrócić uwagę na informacje zawarte w poniższych sekcjach.

- “Nazwy” na stronie 1162
- “Korzystanie z wywołań MQI” na stronie 1162
- “Deklarowanie stałych” na stronie 1162
- “Określanie nazwy struktury” na stronie 1163
- “Określanie formy struktury” na stronie 1163
- “Sterowanie listingami” na stronie 1163
- “Określanie wartości początkowych dla pól” na stronie 1163

- “Pisanie programów do reenteralnego” na stronie 1164
- “Korzystanie z CEDF” na stronie 1164

Nazwy

Nazwy parametrów w opisach wywołań, a także nazwy pól w opisach struktur są wyświetlane w mieszanym przypadku. W makrach języka asemblera dostarczonych wraz z produktem IBM MQwszystkie nazwy są pisane wielkimi literami.

Korzystanie z wywołań MQI

Interfejs MQI jest interfejsem wywołania, dlatego programy w języku asembler muszą obserwować konwencję o łączu systemu operacyjnego.

W szczególności przed wywołaniem wywołania MQI asembler programów w języku asembler musi wskazywać na rejestr R13 w obszarze składowania co najmniej 18 pełnych wyrazów. Ten obszar składowania udostępnia pamięć masową dla wywołanego programu. Przechowuje on rejestry programu wywołującego przed zniszczeniem ich zawartości, a następnie odtwarza zawartość rejestrów programu wywołującego w zamian.

Uwaga: Jest to ważne w przypadku programów w języku asembler CICS , które używają makra DFHEIENT do konfigurowania ich dynamicznej pamięci masowej, ale które mają przesłonić domyślną wartość DATAREG z R13 na inne rejestry. Gdy interfejs CICS Resource Manager odbiera sterowanie z kodu pośredniczącego, zapisuje bieżącą zawartość rejestrów pod adresem, do którego wskazuje R13 . Niezarezerwowanie obszaru składowania w tym celu daje nieprzewidywalne rezultaty i prawdopodobnie spowoduje zakończenie pracy w produkcie CICS.

Deklarowanie stałych

Większość stałych jest deklarowana jako równa w makrze CMQA.

Jednak następujące stałe nie mogą być zdefiniowane jako wyrównanie, a te nie są uwzględniane podczas wywoływania makra przy użyciu opcji domyślnych:

- MQACT_NONE
- MQCI_NONE
- MQFMT_NONE
- ADMINISTRATOR MQFMT_ADMIN
- MQFMT_COMMAND_1
- MQFMT_COMMAND_2
- MQFMT_DEAD_LETTER_HEADER
- Zdarzenie MQFMT_EVENT
- MQFMT_IMS
- MQFMT_IMS_VAR_STRING
- MQFMT_PCF
- MQFMT_STRING
- MQFMT_TRIGGER
- MQFMT_XMIT_Q_HEADER
- MQMI_NONE

Aby uwzględnić je, należy dodać słowo kluczowe EQUONLY=NO podczas wywoływania makra.

CMQA jest chroniona przed wieloma deklaracją, więc można ją wielokrotnie włączyć. Jednak słowo kluczowe EQUONLY staje się aktywne tylko po pierwszym dołączonym makro.

Określanie nazwy struktury

Aby umożliwić zadeklarowaną więcej niż jedną instancję struktury, makro generujące strukturę prefikuje nazwę każdego pola z łańcuchem określanym przez użytkownika i znakiem podkreślenia (_).

Określ łańcuch w momencie wywołania 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_"
```

W deklaracjach struktury w sekcji Opisy wywołań wyświetlany jest domyślny przedrostek.

Określanie formy struktury

Makra mogą generować deklaracje struktury w jednej z dwóch form, sterowanych za pomocą parametru DSECT:

DSECT = TAK
Instrukcja DSECT w języku asemblera jest używana do uruchamiania nowej sekcji danych. Definicja struktury jest natychmiast zgodna z instrukcją DSECT. Pamięć masowa nie jest przydzielona, więc inicjowanie nie jest możliwe. Etykieta w wywołaniu makra jest używana jako nazwa sekcji danych. Jeśli nie określono żadnej etykiety, używana jest nazwa struktury.

DSECT = NIE
Instrukcje DC języka asemblera są używane do definiowania struktury w bieżącej pozycji w podprogramie. Pola są inicjowane z wartościami, które można określić, kodując odpowiednie parametry w wywołaniu makra. Pola, dla których nie są określone żadne wartości w wywołaniu makra, są inicjowane z wartościami domyślnymi.

Parametr DSECT = NO jest przyjmowany, jeśli parametr DSECT nie został określony.

Sterowanie listingami

Istnieje możliwość sterowania wyglądem deklaracji struktury w listingu asemblera za pomocą parametru LIST:

LIST = TAK
Deklaracja struktury pojawia się na liście języków asemblera.

LISTA = NIE
Deklaracja struktury nie jest wyświetlana na liście języka asemblera. Przyjmuje się, że parametr LIST nie jest określony.

Określanie wartości początkowych dla pól

Można określić wartość, która ma być używana do inicjowania pola w strukturze, kodując nazwę tego pola (bez przedrostka) jako parametr w wywołaniu makra, wraz z wymaganą wartością.

Na przykład, aby zadeklarować strukturę deskryptora komunikatu za pomocą pola *MsgType* zainicjowanego za pomocą MQMT_REQUEST, a pole *ReplyToQ* zainicjowane za pomocą łańcucha MY_REPLY_TO_QUEUE, należy użyć następującego kodu:

```
MY_MQMD    CMQMDA      MSGTYPE=MQMT_REQUEST,       X
REPLYTOQ=MY_REPLY_TO_QUEUE
```

Jeśli jako wartość w wywołaniu makra określono stałą nazwaną (lub equate), należy użyć makra CMQA w celu zdefiniowania stałej nazwanej. Nie należy ujmować pojedynczych znaków cudzysłowu (''), które są łańcuchami znaków.

Pisanie programów do reenteralnego

Produkt IBM MQ używa swoich struktur zarówno dla danych wejściowych, jak i wyjściowych. Jeśli chcesz, aby Twój program pozostał reenteralny:

1. Definiowanie wersji roboczych struktur pamięci masowej jako DSECTs lub definiowanie struktur wstawianych w już zdefiniowanym DSECT. Następnie skopiuj DSECT do pamięci masowej, która jest uzyskiwane przy użyciu:
   - W przypadku programów wsadowych i TSO-makra asemblera pamięci masowej lub GETMAIN z/OS
   - W przypadku systemu CICSkomenda DFHEISTG (roboczy pamięć masowa DSECT) lub komenda EXEC CICS GETMAIN

   Aby poprawnie zainicjować te robocze struktury pamięci masowej, należy skopiować stałą wersję odpowiedniej struktury do roboczej wersji pamięci masowej.

   Uwaga: Struktury MQMD i MQXQH mają długość większą niż 256 bajtów. Aby skopiować te struktury do pamięci masowej, należy skorzystać z instrukcji asemblera MVCL.
2. Rezerwuj miejsce w pamięci masowej, korzystając z formularza LIST (MF=L) makra CALL. Jeśli makro CALL jest używane do wywołania MQI, należy użyć formatu EXECUTE (MF=E) makra, korzystając z zarezerwowanej wcześniej pamięci masowej, jak to pokazano w przykładzie w sekcji “Korzystanie z CEDF” na stronie 1164. Więcej przykładów na to, jak to zrobić, można znaleźć w przykładowych programach języka asemblera dostarczonych wraz z produktem IBM MQ.

Użyj opcji RENT języka asemblera, aby określić, czy program jest ponownie rozbawialny.

Aby uzyskać więcej informacji na temat pisania programów, które można przeenterować, patrz *z/OS MVS Application Development Guide: Assembler Language Programs*.

Korzystanie z CEDF

Aby użyć transakcji dostarczonej z produktem CICS, funkcji CEDF (CICS Execution Diagnostic Facility), aby ułatwić debugowanie programu, należy dodać słowo kluczowe ,VL do każdej instrukcji CALL , na przykład:

```
CALL MQCONN,(NAME,HCONN,COMPCODE,REASON),MF=(E,PARMAREA),VL
```

Poprzedni przykład to kod języka asemblera, który jest ponownie dostępny, gdzie PARMAREA jest obszarem w określonej pamięci roboczej, który został określony.

Korzystanie z wywołań MQI

Interfejs MQI jest interfejsem wywołania, dlatego programy w języku asembler muszą obserwować konwencję o łączu systemu operacyjnego. W szczególności przed wywołaniem wywołania MQI asembler programów w języku asembler musi wskazywać na rejestr R13 w obszarze składowania co najmniej 18 pełnych wyrazów. Ten obszar składowania udostępnia pamięć masową dla wywołanego programu. Przechowuje on rejestry programu wywołującego przed zniszczeniem ich zawartości, a następnie odtwarza zawartość rejestrów programu wywołującego w zamian.

Uwaga: Jest to ważne w przypadku programów w języku asembler CICS , które używają makra DFHEIENT do konfigurowania ich dynamicznej pamięci masowej, ale które mają przesłonić domyślną wartość DATAREG z R13 na inne rejestry. Gdy interfejs CICS Resource Manager odbiera sterowanie z kodu pośredniczącego, zapisuje bieżącą zawartość rejestrów pod adresem, do którego wskazuje R13 . Niezarezerwowanie odpowiedniego obszaru składowania w tym celu daje nieprzewidywalne rezultaty i prawdopodobnie spowoduje zakończenie pracy w produkcie CICS.

IBM i Kodowanie programów IBM MQ w języku RPG (tylkoIBM i)

W dokumentacji produktu IBM MQ parametry wywołań, nazwy typów danych, pola struktur i nazwy stałych są opisane za pomocą ich długich nazw. W języku RPG nazwy te są skrócone do sześciu lub mniejszej liczby wielkich liter.

Na przykład pole *MsgType* staje się *MDMT* w języku RPG. Więcej informacji na ten temat zawiera publikacja Skorowidz programistyczny aplikacji IBM i (ILE/RPG).

Kodowanie w języku PL/I (tylko w wersjiz/OS)

Przydatne informacje podczas kodowania dla produktu IBM MQ w PL/I.

Struktury

Struktury są deklarowane za pomocą atrybutu BASED, a więc nie zajmują żadnej pamięci masowej, chyba że program deklaruje co najmniej jedną instancję struktury.

Instancja struktury może zostać zadeklarowana 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 za pomocą atrybutu 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 przypadku języka PL/I nie jest rozróżniana wielkość liter, a więc nazwy wywołań, pól struktury i stałych można zakodować małymi literami, wielkimi literami lub literami o różnej wielkości.

Stałe nazwane

Stałe nazwane są deklarowane jako zmienne makra; w wyniku tego nazwane stałe, które nie są przywoływane przez program, nie zajmują żadnej pamięci w skompilowanej procedurze.

Jednak opcja kompilatora, która powoduje, że źródło jest przetwarzane przez preprocesor makra, 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 danych po podstawieniu zmiennych makra przez procesor makra, 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 produktu IBM MQ

Te przykładowe programy są zapisywane w językach proceduralnych i demonstrują typowe zastosowania interfejsu kolejki komunikatów (Message Queue Interface-MQI). Programy IBM MQ na różnych platformach.

O tym zadaniu

Istnieją dwa zestawy przykładów:

- Przykładowe programy dla systemów rozproszonych i IBM i.
- Przykładowe programy dla produktu z/OS.

Procedura

- Aby dowiedzieć się więcej na temat przykładowych programów, należy użyć poniższych odsyłaczy:
  - “Korzystanie z przykładowych programów na platformie Multiplatforms” na stronie 1166
  - z/OS “Korzystanie z przykładowych programów dla produktu z/OS” na stronie 1272

Pojęcia pokrewne

“Pojęcia związane z projektowaniem aplikacji” na stronie 7
Do pisania aplikacji IBM MQ można użyć wyboru języków proceduralnych lub obiektowych. Przed rozpoczęciem projektowania i pisania aplikacji produktu IBM MQ należy zapoznać się z podstawowymi pojęciami dotyczącymi produktu IBM MQ .

“Tworzenie aplikacji dla składnika IBM MQ” na stronie 5
Istnieje możliwość tworzenia aplikacji w celu wysyłania i odbierania komunikatów oraz do zarządzania menedżerami kolejek i powiązanymi zasobami. Produkt IBM MQ obsługuje aplikacje napisane w wielu różnych językach i w różnych ramach.

“Uwagi dotyczące projektowania aplikacji produktu IBM MQ” na stronie 50
Po zdecydowaniu, w jaki sposób aplikacje mogą korzystać z platform i środowisk, które są dostępne dla użytkownika, należy zdecydować, w jaki sposób korzystać z funkcji oferowanych przez produkt IBM MQ.

“Pisanie aplikacji proceduralnej w celu kolejkowania” na stronie 810
Ta sekcja zawiera informacje na temat pisania aplikacji kolejkowania, łączenia się i rozłączania z menedżerem kolejek, publikowania/subskrypcji oraz obiektów otwierających i zamykających.

“Pisanie aplikacji proceduralnych klienta” na stronie 1008
Co należy wiedzieć, aby pisać aplikacje klienckie w systemie IBM MQ , korzystając z języka proceduralnego.

“Zapisywanie aplikacji publikowania/subskrypcji” na stronie 901
Rozpocznij pisanie aplikacji publikowania/subskrypcji IBM MQ .

“Budowanie aplikacji proceduralnej” na stronie 1099
Aplikację IBM MQ można napisać w jednym z kilku języków proceduralnych, a następnie uruchomić aplikację na kilku różnych platformach.

“Obsługa proceduralnych błędów programu” na stronie 1145
Te informacje wyjaśniają błędy związane z wywołaniami MQI aplikacji, gdy wywołuje połączenie, lub gdy jego komunikat jest dostarczany do miejsca docelowego.

Multi Korzystanie z przykładowych programów na platformie Multiplatforms

Te przykładowe programy proceduralne są dostarczane wraz z produktem. Przykłady są zapisywane w językach C i COBOL i demonstrują typowe zastosowania interfejsu kolejki komunikatów (Message Queue Interface-MQI).

O tym zadaniu

Przykłady nie mają na celu demonstrować ogólnych technik programowania, dlatego niektóre sprawdzanie błędów, które może być uwzględnione w programie produkcyjnym, jest pomijane.

Kod źródłowy dla wszystkich przykładów jest dostarczany wraz z produktem; źródło to zawiera komentarze, które wyjaśniają techniki kolejkowania komunikatów demonstrowe w programach.

IBM i Informacje na temat programowania w języku RPG można znaleźć w podręczniku IBM i Application Programming Reference (ILE/RPG).

Nazwy przykładów rozpoczynają się od przedrostka `amq`. Czwarty znak wskazuje język programowania i kompilator, w razie potrzeby:

- s: język C

- 0: Język COBOL na kompilatorach IBM i Micro Focus
- i: język COBOL tylko w kompilatorach IBM
- m: język COBOL tylko w kompilatorach Micro Focus

Ósmy znak kodu wykonywalnego wskazuje, czy próbka działa w trybie powiązania lokalnego, czy w trybie klienta. Jeśli nie ma ósmego znaku, to próbka działa w trybie powiązań lokalnych. Jeśli ósmy znak jest 'c', to próbka jest uruchamiana w trybie klienta.

Przed uruchomieniem przykładowych aplikacji należy najpierw utworzyć i skonfigurować menedżer kolejek. Informacje na temat konfigurowania menedżera kolejek w celu akceptowania połączeń klienckich zawiera sekcja “Konfigurowanie menedżera kolejek w celu akceptowania połączeń klientów na wielu platformach” na stronie 1177.

Procedura

- Aby dowiedzieć się więcej na temat przykładowych programów, należy użyć poniższych odsyłaczy:
  - “Funkcje demonstruje w przykładowych programach na Multiplatforms” na stronie 1168
  - “Programy przykładowe publikowania/subskrypcji” na stronie 1207
  - “Przykładowe programy umieszczania” na stronie 1212
  - “Przykładowy program listy dystrybucyjnej” na stronie 1198
  - “Przeglądanie przykładowych programów” na stronie 1186
  - “Przykładowy program przeglądarki” na stronie 1187
  - “Pobieranie przykładowych programów” na stronie 1200
  - “Przykładowe programy Message Message” na stronie 1214
  - “Przykładowe programy żądania” na stronie 1222
  - “Przykładowe programy Inquire” na stronie 1205
  - “Właściwości Inquire przykładowego programu Message Handle” na stronie 1207
  - “Przykładowe programy” na stronie 1227
  - “Programy przykładowe Echo” na stronie 1199
  - “Przykładowy program do konwersji danych” na stronie 1190
  - “Przykładowe programy wyzwalające” na stronie 1231
  - “Przykładowy program do umieszczania w pamięci asynchronicznej” na stronie 1185
  - “Przykłady koordynacji bazy danych” na stronie 1191
  - “Przykład transakcji CICS” na stronie 1189
  - “Korzystanie z przykładów TUXEDO w systemach UNIX i Windows” na stronie 1233
  - “Przykład procedury obsługi kolejki niedostarczanych komunikatów” na stronie 1198
  - “Przykładowy program Connect” na stronie 1189
  - “Przykładowy program obsługi wyjścia funkcji API” na stronie 1183
  - “Korzystanie z wyjścia zabezpieczeń SSPI w systemie Windows” na stronie 1246
  - “Uruchamianie przykładów przy użyciu kolejek zdalnych” na stronie 1247
  - “Przykładowy program monitorowania kolejki klastra (AMQSCLM)” na stronie 1247
  - “Przykładowy program do wyszukiwania punktów końcowych połączenia (CEPL)” na stronie 1257

Pojęcia pokrewne

“Przykładowe programy w języku C++” na stronie 540
Dostarczane są cztery przykładowe programy demonstrujące pobieranie i umieszczanie komunikatów.

Zadania pokrewne

“Korzystanie z przykładowych programów dla produktu z/OS” na stronie 1272

Przykładowe aplikacje proceduralne dostarczane wraz z produktem IBM MQ for z/OS demonstrują typowe zastosowania interfejsu kolejki komunikatów (Message Queue Interface-MQI).

Multi *Funkcje demonstruje w przykładowych programach na Multiplatforms*

Kolekcja tabel, w których przedstawiono techniki pokazanego przez programy przykładowe produktu IBM MQ .

Wszystkie przykładowe kolejki są otwierane i zamykane przy użyciu wywołań MQOPEN i MQCLOSE, dlatego te techniki nie są wymienione oddzielnie w tabelach. Zapoznaj się z nagłówkiem, w którym znajduje się interesowana platforma.

z/OS Informacje na temat platformy z/OS zawiera sekcja "Korzystanie z przykładowych programów dla produktu z/OS" na stronie 1272.

Linux UNIX *Przykłady dla systemów UNIX and Linux*

Techniki pokazanego przez programy przykładowe dla IBM MQ w systemach UNIX and Linux .

Aby dowiedzieć się, gdzie są przechowywane przykładowe programy dla systemu IBM MQ w systemach UNIX and Linux , należy zapoznać się z "Przygotowywanie i uruchamianie przykładowych programów w systemie UNIX and Linux" na stronie 1180 .

Tabela 163 na stronie 1168 Tabela zawiera listę dostępnych plików źródłowych języka C i języka COBOL oraz informacje o tym, czy plik wykonywalny serwera czy klienta jest dołączony.

Tabela 163. Przykładowe programy, które demonstrują użycie interfejsu MQI (C i COBOL) w systemie UNIX and Linux.

Tabela z czterema kolumnami. W pierwszych kolumnach znajduje się lista technik pokazanych przez przykłady. W drugiej kolumnie znajduje się lista przykładów języka C, a trzecia-przykłady w języku COBOL, które demonstrują każdą z technik wymienionych w pierwszej kolumnie. Czwarta kolumna wskazuje, czy plik wykonywalny serwera C jest lub nie jest dołączony, a piąta kolumna wskazuje, czy plik wykonywalny klienta C jest lub nie jest dołączony.

Technika	**C (źródło) ("1" na stronie 1171)**	**COBOL (źródło) ("2" na stronie 1171)**	**Serwer (kod wykonywalny C)**	**Klient (kod wykonywalny C)**
Korzystanie z interfejsu publikowania/ subskrybowania	amqspuba amqssuba amqssbxa	brak próby	amqspub amqssub amqssbx	brak próby
Umieszczanie komunikatów przy użyciu wywołania MQPUT	amqsput0	amq0put0	amqsput	amqsputc
Umieszczanie pojedynczego komunikatu przy użyciu wywołania MQPUT1	amqsinqa amqsecha	amqminqx amqmechx amqiinqx amqiechx	amqsinq amqsech	amqsechc
Umieszczanie komunikatów na liście dystrybucyjnej ("3" na stronie 1171)	amqsptl0	amq0ptl0.cbl	amqsptl	amqsptlc
Odpowiadanie na komunikat żądania	amqsinqa	amqminqx amqiinqx	amqsinq	brak próby
Pobieranie komunikatów za pomocą przeglądania (bez oczekiwania)	amqsgbr0	amq0gbr0	amqsgbr	brak próby
Pobieranie komunikatów (czekaj z limitem czasu)	amqsget0	amq0get0	amqsget	amqsgetc

Tabela 163. Przykładowe programy, które demonstrują użycie interfejsu MQI (C i COBOL) w systemie UNIX and Linux.

Tabela z czterema kolumnami. W pierwszych kolumnach znajduje się lista technik pokazanych przez przykłady. W drugiej kolumnie znajduje się lista przykładów języka C, a trzecia-przykłady w języku COBOL, które demonstrują każdą z technik wymienionych w pierwszej kolumnie. Czwarta kolumna wskazuje, czy plik wykonywalny serwera C jest lub nie jest dołączony, a piąta kolumna wskazuje, czy plik wykonywalny klienta C jest lub nie jest dołączony.

(kontynuacja)

Technika	**C (źródło) ("1" na stronie 1171)**	**COBOL (źródło) ("2" na stronie 1171)**	**Serwer (kod wykonywalny C)**	**Klient (kod wykonywalny C)**
Pobieranie komunikatów (nieograniczony czas oczekiwania)	amqstrg0	brak próby	amqstrg	amqstrgc
Pobieranie komunikatów (z konwersją danych)	amqsecha	brak próby	amqsech	brak próby
Umieszczanie komunikatów referencyjnych w kolejce ("3" na stronie 1171)	amqsprma	brak próby	amqsprm	amqsprmc
Pobieranie komunikatów referencyjnych z kolejki ("3" na stronie 1171)	amqsgrma	brak próby	amqsgrm	amqsgrmc
Wyjście odwołania kanału komunikatów ("3" na stronie 1171)	amqsqrma amqsxrma	brak próby	amqsxrm	brak próby
Przeglądanie pierwszych 20 znaków wiadomości	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Przeglądanie pełnych komunikatów	amqsbcg0	brak próby	amqsbcg	amqsbcgc
Korzystanie z 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 próby
Korzystanie z wywołania MQSET	amqsseta	amqmsetx amqisetx	amqsset	amqssetc
Korzystanie z kolejki odpowiedzi	amqsreq0	amq0req0	amqsreq	amqsreqc
Żądanie wyjątków komunikatów	amqsreq0	amq0req0	amqsreq	brak próby
Akceptowanie obciętej wiadomości	amqsgbr0	amq0gbr0	amqsgbr	brak próby
Korzystanie z przetłumaczanej nazwy kolejki	amqsgbr0	amq0gbr0	amqsgbr	brak próby
Wyzwalanie procesu	amqstrg0	brak próby	amqstrg	amqstrgc
Używanie konwersji danych	("4" na stronie 1171)	brak próby	brak próby	brak próby
IBM MQ (koordynowanie menedżerów baz danych zgodnych z interfejsem XA) z dostępem do jednej bazy danych przy użyciu języka SQL	amqsxas0.sqc Db2 amqsxas0.ec Informix	amq0xas0.sqb	brak próby	brak próby

Tabela 163. Przykładowe programy, które demonstrują użycie interfejsu MQI (C i COBOL) w systemie UNIX and Linux.

Tabela z czterema kolumnami. W pierwszych kolumnach znajduje się lista technik pokazanych przez przykłady. W drugiej kolumnie znajduje się lista przykładów języka C, a trzecia-przykłady w języku COBOL, które demonstrują każdą z technik wymienionych w pierwszej kolumnie. Czwarta kolumna wskazuje, czy plik wykonywalny serwera C jest lub nie jest dołączony, a piąta kolumna wskazuje, czy plik wykonywalny klienta C jest lub nie jest dołączony.

(kontynuacja)

Technika	**C (źródło) ("1" na stronie 1171)**	**COBOL (źródło) ("2" na stronie 1171)**	**Serwer (kod wykonywalny C)**	**Klient (kod wykonywalny C)**
IBM MQ (koordynowanie menedżerów baz danych zgodnych z XA) z dostępem do dwóch baz danych przy użyciu języka SQL	amqsxag0.c amqsxab0.sqc amqsxaf0.sqc	amq0xag0.cbl amq0xab0.sqb amq0xaf0.sqb	brak próby	brak próby
Transakcja CICS ("5" na stronie 1171)	amqscic0.ccs	brak próby	amqscic0	brak próby
Transakcja encina ("3" na stronie 1171)	amqsxae0	brak próby	amqsxae0	brak próby
Transakcja TUXEDO w celu umieszczania komunikatów "6" na stronie 1171)	amqstxpx	brak próby	brak próby	brak próby
Transakcja TUXEDO w celu pobrania komunikatów ("6" na stronie 1171)	amqstxgx	brak próby	brak próby	brak próby
Serwer dla TUXEDO ("6" na stronie 1171)	amqstxsx	brak próby	brak próby	brak próby
procedura obsługi kolejki niedostarczonych komunikatów	Katalog `./tools/c/Samples/dlq` ("7" na stronie 1171)	brak próby	amqsdlq	brak próby
Z poziomu klienta MQI: umieszczanie komunikatu	brak próby	brak próby	brak próby	amqsputc
Z poziomu klienta MQI: uzyskiwanie komunikatu	brak próby	brak próby	brak próby	amqsgetc
Nawiąże połączenie z menedżerem kolejek przy użyciu produktu MQCONNX	amqscnxc	brak próby	brak próby	amqscnxc
Korzystanie z wyjść funkcji API	amqsaxe0	brak próby	amqsaxe	brak próby
Wyjście równoważenia obciążenia klastra	amqswlm0	brak próby	amqswlm	brak próby
Asynchroniczne umieszczanie komunikatów i pobieranie statusu za pomocą wywołania MQSTAT	amqsapt0	brak próby	amqsapt	amqsaptc
Klienty z możliwością ponownego nawiązania połączenia	amqsphac amqsghac amqsmhac	brak próby	bez zastosowania	amqsphac amqsghac amqsmhac
Używanie konsumentów komunikatów do asynchronicznego korzystania z komunikatów z wielu kolejek	amqscbf0	brak próby	amqscbf	amqscbfc

Tabela 163. Przykładowe programy, które demonstrują użycie interfejsu MQI (C i COBOL) w systemie UNIX and Linux.

Tabela z czterema kolumnami. W pierwszych kolumnach znajduje się lista technik pokazanych przez przykłady. W drugiej kolumnie znajduje się lista przykładów języka C, a trzecia-przykłady w języku COBOL, które demonstrują każdą z technik wymienionych w pierwszej kolumnie. Czwarta kolumna wskazuje, czy plik wykonywalny serwera C jest lub nie jest dołączony, a piąta kolumna wskazuje, czy plik wykonywalny klienta C jest lub nie jest dołączony.

(kontynuacja)

Technika	**C (źródło) ("1" na stronie 1171)**	**COBOL (źródło) ("2" na stronie 1171)**	**Serwer (kod wykonywalny C)**	**Klient (kod wykonywalny C)**
Określanie informacji o połączeniu TLS w tabeli MQCONNX	amqssslc	brak próby	bez zastosowania	amqssslc

Uwagi:

1. Wersja wykonywalna przykładów produktu IBM MQ MQI client współużytkuje to samo źródło, co przykłady, które działają w środowisku serwera.
2. Skompiluj programy zaczynające się od 'amqm' z kompilatorem Micro Focus COBOL, które rozpoczynają się od 'amqi' z kompilatorem języka COBOL IBM , a także rozpoczynające się od 'amq0'.
3. UNIX Obsługiwane tylko w systemach IBM MQ for AIX i IBM MQ for Solaris .
4. UNIX W systemach IBM MQ for AIX, i IBM MQ for Solaris ten program ma nazwę `amqsvfc0.c`
5. AIX Produkt CICS jest obsługiwany tylko przez produkty IBM MQ for AIX .
6. Linux TUXEDO is not supported by IBM MQ for Linux on System p.
7. Źródło dla procedury obsługi kolejki niedostarczonych komunikatów składa się z kilku plików i znajduje się w oddzielnym katalogu.

Szczegółowe informacje na temat obsługi systemów UNIX and Linux można znaleźć w sekcji Wymagania systemowe produktu IBM MQ.

Windows *Przykłady dla produktu IBM MQ for Windows*

Techniki pokazanego przez programy przykładowe dla produktu IBM MQ for Windows.

Tabela 164 na stronie 1171 zawiera listę plików źródłowych C i COBOL, które są udostępniane oraz informacje o tym, czy jest dostępny plik wykonywalny serwera czy klienta.

Tabela 164. Programy przykładowe IBM MQ for Windows demonstruje użycie interfejsu MQI (C i COBOL)

Technika	**C (źródło)**	**COBOL (źródło)**	**Serwer (kod wykonywalny C)**	**Klient (kod wykonywalny C)**
Korzystanie z interfejsu publikowania/ subskrybowania	amqspuba amqssuba amqssbxa	brak próby	amqspub amqssub amqssbx	brak próby
Umieszczanie komunikatów przy użyciu wywołania MQPUT	amqsput0	amq0put0	amqsput	amqsputc
Umieszczanie pojedynczego komunikatu przy użyciu wywołania MQPUT1	amqsinqa amqsecha	amqminq2 amqmech2 amqiinq2 amqiech2	amqsinq amqsech	amqsinqc amqsechc

Tabela 164. Programy przykładowe IBM MQ for Windows demonstruje użycie interfejsu MQI (C i COBOL) (kontynuacja)

Technika	**C (źródło)**	**COBOL (źródło)**	**Serwer (kod wykonywalny C)**	**Klient (kod wykonywalny C)**
Umieszczanie komunikatów na liście dystrybucyjnej	amqsptl0	amq0ptl0.cbl	amqsptl	amqsptlc
Odpowiadanie na komunikat żądania	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
Pobieranie komunikatów (bez oczekiwania)	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Pobieranie komunikatów (czekaj z limitem czasu)	amqsget0	amq0get0	amqsget	amqsgetc
Pobieranie komunikatów (nieograniczony czas oczekiwania)	amqstrg0	brak próby	amqstrg	amqstrgc
Pobieranie komunikatów (z konwersją danych)	amqsecha	brak próby	amqsech	amqsechc
Umieszczanie komunikatów odniesienia w kolejce	amqsprma	brak próby	amqsprm	amqsprmc
Pobieranie komunikatów odwołania z kolejki	amqsgrma	brak próby	amqsgrm	amqsgrmc
Wyjście odwołania kanału komunikatów	amqsqrma amqsxrma	brak próby	amqsxrm	brak próby
Przeglądanie pierwszych 20 znaków wiadomości	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Przeglądanie pełnych komunikatów	amqsbcg0	brak próby	amqsbcg	amqsbcgc
Korzystanie z współużytkowanej kolejki wejściowej	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
Korzystanie z wyłącznej kolejki wejściowej	amqstrg0	amq0req0	amqstrg	amqstrgc
Korzystanie z wywołania MQINQ	amqsinqa	amqminq2 amqiinq2	amqsinq	amqsinqc
Korzystanie z wywołania MQSET	amqsseta	amqmset2 amqiset2	amqsset	amqssetc
Korzystanie z wywołania MQINQMP	amqsiqma	brak próby	brak próby	brak próby
Korzystanie z kolejki odpowiedzi	amqsreq0	amq0req0	amqsreq	amqsreqc
Żądanie wyjątków komunikatów	amqsreq0	amq0req0	amqsreq	amqsreqc
Akceptowanie obciętej wiadomości	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Korzystanie z przetłumaczanej nazwy kolejki	amqsgbr0	amq0gbr0	amqsgbr	amqsgbrc
Wyzwalanie procesu	amqstrg0	brak próby	amqstrg	amqstrgc
Używanie konwersji danych	amqsvfc0	brak próby	brak próby	brak próby
IBM MQ (koordynowanie menedżerów baz danych zgodnych z interfejsem XA) z dostępem do jednej bazy danych przy użyciu języka SQL	amqsxas0.sqc Db2 amqsxas0.ec Informix	amq0xas0.sqb	brak próby	brak próby

Tabela 164. Programy przykładowe IBM MQ for Windows demonstruje użycie interfejsu MQI (C i COBOL) (kontynuacja)

Technika	**C (źródło)**	**COBOL (źródło)**	**Serwer (kod wykonywalny C)**	**Klient (kod wykonywalny C)**
IBM MQ (koordynowanie menedżerów baz danych zgodnych z XA) z dostępem do dwóch baz danych przy użyciu języka SQL	amqsxag0.c amqsxab0.sqc Db2 amqsxaf0.sqc Db2	amq0xag0.cbl amq0xab0.sqb amq0xaf0.sqb	brak próby	brak próby
Transakcja TUXEDO w celu umieszczania komunikatów	amqstxpx	brak próby	brak próby	brak próby
Transakcja TUXEDO w celu pobrania komunikatów	amqstxgx	brak próby	brak próby	brak próby
Serwer dla TUXEDO	amqstxsx	brak próby	brak próby	brak próby
procedura obsługi kolejki niedostarczonych komunikatów	Katalog `./tools/c/Samples/dlq` (“1” na stronie 1173)	brak próby	amqsdlq	brak próby
Z IBM MQ MQI client, umieszczając komunikat	brak próby	brak próby	brak próby	amqsputc
Uzyskiwanie komunikatu z poziomu IBM MQ MQI client	brak próby	brak próby	brak próby	amqsgetc
Nawiąże połączenie z menedżerem kolejek przy użyciu produktu MQCONNX	amqscnxc	brak próby	brak próby	amqscnxc
Korzystanie z wyjść funkcji API	amqsaxe0	brak próby	amqsaxe	brak próby
Równoważenie obciążenia klastra	amqswlm0	brak próby	amqswlm	brak próby
Procedury bezpieczeństwa SSPI	amqsspin	brak próby	amqrspin.dll	amqrspin.dll
Asynchroniczne umieszczanie komunikatów i pobieranie statusu za pomocą wywołania MQSTAT	amqsapt0	brak próby	amqsapt	amqsaptc
Klienty z możliwością ponownego nawiązania połączenia	amqsphac amqsghac amqsmhac	brak próby	Nie dotyczy	amqsphac amqsghac amqsmhac
Używanie konsumentów komunikatów do asynchronicznego korzystania z komunikatów z wielu kolejek	amqscbf0	brak próby	amqscbf	amqscbfc
Określanie informacji o połączeniu TLS w tabeli MQCONNX	amqsssslc	brak próby	bez zastosowania	amqsssslc

Uwagi:

1. Źródło dla procedury obsługi kolejki niedostarczonych komunikatów składa się z kilku plików i znajduje się w oddzielnym katalogu.

Windows *Przykłady Visual Basic dla produktu IBM MQ for Windows*

Techniki pokazanego przez programy przykładowe dla IBM MQ w systemach Windows .

Tabela 165 na stronie 1174 przedstawia techniki pokazanego przez programy przykładowe programu IBM MQ for Windows .

Projekt może zawierać kilka plików. Po otwarciu projektu w języku Visual Basic, pozostałe pliki są ładowane automatycznie. Nie są dostępne żadne programy wykonywalne.

Wszystkie przykładowe projekty, z wyjątkiem mqtrivc.vbp, są skonfigurowane do pracy z serwerem IBM MQ . Aby dowiedzieć się, jak zmienić przykładowe projekty, aby pracować z klientami IBM MQ , należy zapoznać się z “Przygotowywanie programów Visual Basic w programie Windows” na stronie 1126.

Tabela 165. Programy przykładowe IBM MQ for Windows demonstruje użycie interfejsu MQI (Visual Basic)

Technika	**Nazwa pliku projektu**
Umieszczanie komunikatów przy użyciu wywołania MQPUT	amqsputb.vbp
Pobieranie komunikatów za pomocą wywołania MQGET	amqsgetb.vbp
Przeglądanie kolejki przy użyciu wywołania MQGET	amqsbcgb.vbp
Prosty przykład MQGET i MQPUT (klient)	mqtrivc.vbp
Prosty przykład MQGET i MQPUT (serwer)	mqtrivs.vbp
Umieszczanie i pobieranie łańcuchów i struktur zdefiniowanych przez użytkownika za pomocą 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 za pomocą interfejsu MQAI	amqsaiem.vbp

IBM i *Przykłady dla produktu IBM i*

Techniki pokazanego przez programy przykładowe dla IBM MQ w systemach IBM i .

Tabela 166 na stronie 1174 przedstawia techniki pokazanego przez programy przykładowe programu IBM MQ for IBM i . Niektóre techniki występują w więcej niż jednym programie przykładowym, ale w tabeli znajduje się tylko jeden program.

Tabela 166. Przykładowe programy demonstruje użycie interfejsu MQI (C i COBOL) w systemie IBM i

Technika	**C (źródło) (“1” na stronie 1176)**	**COBOL (źródło) (“2” na stronie 1176)**	**RPG (źródło) (“3” na stronie 1176)**	**Klient (kod wykonywalny C) (4)**
Umieszczanie komunikatów przy użyciu wywołania MQPUT	AMQSPUT0	AMQ0PUT4	AMQ3PUT4	amqsputc
Umieszczanie komunikatów z pliku danych przy użyciu wywołania MQPUT	AMQSPUT4	brak próby	brak próby	brak próby
Umieszczanie pojedynczego komunikatu przy użyciu wywołania MQPUT1	AMQSINQ4, AMQSECH4	AMQ0INQ4, AMQ0ECH4	AMQ3INQ4, AMQ3ECH4	AMQSINQC, AMQSECHC
Umieszczanie komunikatów na liście dystrybucyjnej	AMQSPTL4	brak próby	brak próby	AMQSPTLC
Odpowiadanie na komunikat żądania	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC

Tabela 166. Przykładowe programy demonstruje użycie interfejsu MQI (C i COBOL) w systemie IBM i (kontynuacja)

Technika	**C (źródło) ("1" na stronie 1176)**	**COBOL (źródło) ("2" na stronie 1176)**	**RPG (źródło) ("3" na stronie 1176)**	**Klient (kod wykonywalny C) (4)**
Pobieranie komunikatów (bez oczekiwania)	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
Pobieranie komunikatów (czekaj z limitem czasu)	AMQSGET4	AMQ0GET4	AMQ3GET4	AMQSGETC
Pobieranie komunikatów (nieograniczony czas oczekiwania)	AMQSTRG4	brak próby	AMQ3TRG4	AMQSTRGC
Pobieranie komunikatów (z konwersją danych)	AMQSECH4	AMQ0ECH4	AMQ3ECH4	AMQSECHC
Umieszczanie komunikatów odniesienia w kolejce	AMQSPRM4	brak próby	brak próby	Kontekst monitorowania AMQSPRMC
Pobieranie komunikatów odwołania z kolejki	AMQSGRM4	brak próby	brak próby	AMQSGRMC
Wyjście odwołania kanału komunikatów	AMQSQRM4, AMQSXRM4	brak próby	brak próby	brak próby
Wyjście komunikatu	AMQSCMX4	brak próby	brak próby	brak próby
Przeglądanie pierwszych 49 znaków wiadomości	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
Przeglądanie pełnych komunikatów	AMQSBCG4	brak próby	brak próby	AMQSBCGC
Korzystanie z współużytkowanej kolejki wejściowej	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
Korzystanie z wyłącznej kolejki wejściowej	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
Korzystanie z wywołania MQINQ	AMQSINQ4	AMQ0INQ4	AMQ3INQ4	AMQSINQC
Korzystanie z wywołania MQSET	AMQSSET4	AMQ0SET4	AMQ3SET4	AMQSSETC
Korzystanie z kolejki odpowiedzi	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
Żądanie wyjątków komunikatów	AMQSREQ4	AMQ0REQ4	AMQ3REQ4	AMQSREQC
Akceptowanie obciętej wiadomości	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
Korzystanie z przetłumaczanej nazwy kolejki	AMQSGBR4	AMQ0GBR4	AMQ3GBR4	AMQSGBRC
Wyzwalanie procesu	AMQSTRG4	brak próby	AMQ3TRG4	AMQSTRGC
Serwer wyzwalacza	AMQSERV4	brak próby	AMQ3SRV4	brak próby
Korzystanie z serwera wyzwalaczy (w tym transakcji produktu CICS)	AMQSERV4	brak próby	AMQ3SRV4	brak próby
Używanie konwersji danych	AMQSVFC4	brak próby	brak próby	brak próby
Korzystanie z wyjść funkcji API	AMQSAXE0	brak próby	brak próby	brak próby
Równoważenie obciążenia klastra	AMQSWLM0	brak próby	brak próby	brak próby
Asynchroniczne umieszczanie komunikatów i pobieranie statusu za pomocą wywołania MQSTAT	AMQSAPT0	brak próby	brak próby	AMQSAPTC

Tabela 166. Przykładowe programy demonstruje użycie interfejsu MQI (C i COBOL) w systemie IBM i (kontynuacja)				
Technika	**C (źródło) ("1" na stronie 1176)**	**COBOL (źródło) ("2" na stronie 1176)**	**RPG (źródło) ("3" na stronie 1176)**	**Klient (kod wykonywalny C) (4)**
Korzystanie z interfejsu publikowania/ subskrybowania	AMQSPUBA, AMQSSUBA, AMQSSBXA	brak próby	brak próby	AMQSPUBC, AMQSSUBC, AMQSSBXC
Klienty z możliwością ponownego połączenia (5)	AMQSPHAC, AMQSGHAC, AMQSMHAC	brak próby	brak próby	brak próby
Korzystanie z konsumentów komunikatów w celu asynchronicznego wykorzystania komunikatów z wielu kolejek (5)	AMQSCBFO	brak próby	brak próby	brak próby
Określanie informacji o połączeniu TLS w tabeli MQCONNX	AMQSSSLC	brak próby	brak próby	AMQSSSLC
Nawiąże połączenie z menedżerem kolejek przy użyciu produktu MQCONNX	AMQSCNXC	brak próby	brak próby	AMQSCNXC

Uwagi:

1. Kod źródłowy dla przykładów C znajduje się w pliku QMQMSAMP/QCSRC. Pliki włączane istnieją jako elementy w zbiorze QMQM/H.
2. Kod źródłowy dla przykładów języka COBOL znajduje się w plikach QMQMSAMP/QCBLLESRC. Elementy te mają nazwę AMQ0 xxx 4, gdzie xxx wskazuje funkcję przykładową.
3. Kod źródłowy dla przykładów RPG znajduje się w QMQMSAMP/QRPGLESRC. Członkowie mają nazwę AMQ3 xxx 4, gdzie xxx wskazuje funkcję przykładową. Elementy kopii istnieją w QMQM/QRPGLESRC. Każda nazwa elementu ma przyrostek G.
4. Wersja wykonywalna przykładów produktu IBM MQ MQI client współużytkuje to samo źródło, co przykłady, które działają w środowisku serwera. Źródło dla przykładów w środowisku klienta jest takie samo, jak serwer. Przykłady produktu IBM MQ MQI client są powiązane z biblioteką kliencką LIBMQIC, a przykłady serwera IBM MQ są powiązane z biblioteką serwera LIBMQM.
5. Jeśli plik wykonywalny klienta dla przykładowej aplikacji klienta Reconnectable i aplikacji konsumującej asynchronicznie musi zostać uruchomiony, musi być skompilowany i połączony z biblioteką wielowątkową LIBMQIC_R. W związku z tym musi on być uruchamiany w środowisku wielowątkowym. Ustaw zmienną środowiskową QIBM_MULTI_THREADED na wartość 'Y' i uruchom aplikację z aplikacji qsh.

   Więcej informacji na ten temat zawiera sekcja Konfigurowanie produktu IBM MQ z językiem Java i JMS .

Oprócz tych opcji przykładowa opcja IBM MQ for IBM i zawiera przykładowy plik danych, który jest używany jako dane wejściowe do programów przykładowych, AMQSDATA oraz przykładowych programów CL, które demonstrują zadania administracyjne. Przykłady języka CL są opisane w sekcji Administrowanie produktem IBM i . Można użyć przykładowego programu CL `amqsamp4` , aby utworzyć kolejki, które będą używane z przykładowymi programami opisanymi w tym temacie.

Windows IBM i UNIX *Przygotowywanie i uruchamianie przykładowych programów*

Po zakończeniu początkowego przygotowania można następnie uruchomić programy przykładowe.

O tym zadaniu

Przed uruchomieniem programów przykładowych należy najpierw utworzyć menedżer kolejek, a także utworzyć potrzebne kolejki. Może być również konieczne wykonanie dodatkowych czynności przygotowawczego, na przykład w celu uruchomienia przykładów w języku COBOL. Po zakończeniu niezbędnego przygotowania można następnie uruchomić programy przykładowe.

Procedura

Informacje na temat przygotowywania i uruchamiania przykładowych programów znajdują się w następujących tematach:

- "Przygotowywanie i uruchamianie przykładowych programów w systemie IBM i" na stronie 1179
- "Przygotowywanie i uruchamianie przykładowych programów w systemie UNIX and Linux" na stronie 1180
- "Przygotowywanie i uruchamianie przykładowych programów w systemie Windows" na stronie 1182

Windows IBM i UNIX *Konfigurowanie menedżera kolejek w celu akceptowania połączeń klientów na wielu platformach*

Przed uruchomieniem przykładowych aplikacji należy najpierw utworzyć menedżer kolejek. Następnie można skonfigurować menedżer kolejek w taki sposób, aby bezpiecznie akceptować 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, wydając komendę MQSC:

```
DISPLAY QMGR CHLAUTH
```

Ważne: To zadanie oczekuje, że włączone są rekordy uwierzytelniania kanału. Jeśli jest to menedżer kolejek używany przez innych użytkowników i aplikacje, zmiana tego ustawienia będzie mieć wpływ na wszystkich innych użytkowników i aplikacje. Jeśli menedżer kolejek nie korzysta z rekordów uwierzytelniania kanału, krok 4 może zostać zastąpiony alternatywną metodą uwierzytelniania (na przykład wyjście zabezpieczeń), które ustawia wartość MCAUSER na wartość *non-privileged-user-id* (identyfikator użytkownika bez uprawnień uprzywilejowanych), który zostanie uzyskany w kroku "1" na stronie 1177.

Należy wiedzieć, która nazwa kanału ma być używana przez aplikację w taki sposób, aby aplikacja mogła używać kanału. Należy również wiedzieć, które obiekty, na przykład kolejki lub tematy, mają być używane przez aplikację, dzięki czemu aplikacja będzie mogła je używać.

O tym zadaniu

To zadanie tworzy nieuprawnionego ID użytkownika, który ma być używany dla aplikacji klienckiej, która łączy się z menedżerem kolejek. Dostęp do aplikacji klienckiej jest udzielany tylko w celu użycia kanału, którego potrzebuje, oraz kolejki, której potrzebuje, korzystając z tego identyfikatora użytkownika.

Procedura

1. Uzyskaj identyfikator użytkownika w systemie, na którym jest uruchomiony menedżer kolejek. W przypadku tego zadania ten identyfikator użytkownika nie może być uprzywilejowanym użytkownikiem administracyjnym. Ten identyfikator użytkownika będzie miał uprawnienia, w ramach którego połączenie klienta zostanie uruchomione w menedżerze kolejek.
2. Uruchom program nasłuchujący z następującymi komendami, w których:

   nazwa_menedżera_kolejek to nazwa menedżera kolejek
   nnnn jest wybranym numerem portu

a) Windows UNIX

W systemach UNIX i Windows :

```
runmqlsr -t tcp -m qmgr-name -p nnnn
```

b) IBM i

IBM i:

```
STRMQMLSR MQMNAME(qmgr-name) PORT(nnnn)
```

3. Jeśli aplikacja korzysta z systemu SYSTEM.DEF.SVRCONN , a następnie ten kanał jest już zdefiniowany. Jeśli aplikacja używa innego kanału, utwórz ją, wydając 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, zezwalając na korzystanie z kanału tylko przez adres IP systemu klienckiego za pomocą komendy MQSC:

```
SET CHLAUTH(' channel-name ') TYPE(ADDRESSMAP) ADDRESS(' client-machine-IP-address ') +
MCAUSER(' non-privileged-user-id ')
```

nazwa-kanału jest nazwą kanału.

adres_IP jest adresem IP systemu klienckiego użytkownika.

Jeśli przykładowa aplikacja kliencka działa na tym samym komputerze, co menedżer kolejek, należy użyć adresu IP '127.0.0.1', jeśli aplikacja ma nawiązać połączenie przy użyciu 'localhost'. Jeśli zostanie nawiązane połączenie kilku różnych komputerów klienckich, można użyć wzorca lub zakresu zamiast pojedynczego adresu IP. Szczegółowe informacje na ten temat zawiera sekcja Ogólne adresy IP .

identyfikator-użytkownika bez uprawnień jest identyfikatorem użytkownika uzyskanym w kroku “1” na stronie 1177

5. Jeśli aplikacja korzysta z systemu SYSTEM.DEFAULT.LOCAL.QUEUE , a następnie ta kolejka jest już zdefiniowana. Jeśli aplikacja używa innej kolejki, utwórz ją, wydając komendę MQSC:

```
DEFINE QLOCAL(' queue-name ') DESCR('Queue for use by sample programs')
```

nazwa-kolejki jest nazwą kolejki.

6. Przyznaj dostęp do połączenia z menedżerem kolejek i sprawdź, czy menedżer kolejek:

a) Windows IBM i UNIX

W systemach IBM i IBM i, UNIX i Windows wydaj następujące komendy MQSC:

```
SET AUTHREC OBJTYPE(QMGR) PRINCIPAL(' non-privileged-user-id ') +
AUTHADD(CONNECT, INQ)
```

identyfikator-użytkownika bez uprawnień jest identyfikatorem użytkownika uzyskanym w kroku “1” na stronie 1177

7. Jeśli aplikacja jest aplikacją typu punkt z punktem, to znaczy, że korzysta z kolejek, nadając uprawnienia do uzyskiwania informacji oraz umieszczając i pobierając komunikaty przy użyciu kolejki przy użyciu identyfikatora użytkownika, który ma być używany, wydając komendy MQSC:

a) Windows IBM i UNIX

W systemach IBM i IBM i, UNIX i Windows wydaj następujące komendy MQSC:

```
SET AUTHREC PROFILE(' queue-name ') OBJTYPE(QUEUE) +
PRINCIPAL(' non-privileged-user-id ') AUTHADD(PUT, GET, INQ, BROWSE)
```

nazwa-kolejki jest nazwą kolejki.

identyfikator-użytkownika bez uprawnień jest identyfikatorem użytkownika uzyskanym w kroku “1” na stronie 1177

8. Jeśli aplikacja jest aplikacją publikowania/subskrypcji, to znaczy, że korzysta z tematów, przyznaj dostęp, aby zezwolić na publikowanie i subskrybowanie tematu przy użyciu identyfikatora użytkownika, który ma być używany, wydając komendy MQSC:

a) Windows IBM i UNIX

W systemach IBM i IBM i, UNIX i Windows wydaj następujące komendy MQSC:

```
SET AUTHREC PROFILE('SYSTEM.BASE.TOPIC') OBJTYPE(TOPIC) +
PRINCIPAL(' non-privileged-user-id ') AUTHADD(PUB, SUB)
```

identyfikator-użytkownika bez uprawnień jest identyfikatorem użytkownika uzyskanym w kroku “1” na stronie 1177

Spowoduje to nadanie *nieuprawnionego identyfikatora użytkownika* dostępu do dowolnego tematu w drzewie tematów. Można również zdefiniować obiekt tematu przy użyciu produktu **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 połączyć się z menedżerem kolejek i umieścić lub pobrać komunikaty przy użyciu kolejki.

Pojęcia pokrewne

IBM i Uprawnienia IBM MQ w systemie IBM i

Zadania pokrewne

Nadawanie dostępu do obiektu IBM MQ w systemach UNIX lub Linux oraz Windows

Odsyłacze pokrewne

USTAW WARTOŚĆ CHLAUTH
Zdefiniowanie kanału
DEFINE QLOCAL
SET AUTHREC

IBM i *Przygotowywanie i uruchamianie przykładowych programów w systemie IBM i*

Przed uruchomieniem przykładowych programów w systemie IBM inależy najpierw utworzyć menedżer kolejek, a także utworzyć potrzebne kolejki. Aby uruchomić przykłady w języku COBOL, konieczne może być wykonanie dodatkowych czynności przygotowawcze.

O tym zadaniu

Źródło dla przykładowych programów IBM MQ for IBM i jest udostępniane w bibliotece QMQMSAMP jako elementy QCSRC, QCLSRC, QCBLLESRC i QRPGLESRC.

Podczas uruchamiania przykładów można używać własnych kolejek lub można uruchomić przykładowy program AMQSAMP4 w celu utworzenia niektórych kolejek przykładowych. Źródło dla programu AMQSAMP4 jest zawarte w zbiorze QCLSRC w bibliotece QMQMSAMP. Można go skompilować za pomocą komendy CRTCLPGM.

Aby uruchomić przykłady, należy użyć plików wykonywalnych języka C, które są dostarczane w bibliotece QMQM, lub skompilować je w podobny sposób do dowolnej innej aplikacji produktu IBM MQ .

Procedura

1. Utwórz menedżer kolejek i skonfiguruj domyślne definicje.

   Należy to zrobić, aby możliwe było uruchomienie dowolnego z programów przykładowych. Więcej informacji na temat tworzenia menedżera kolejek zawiera sekcja Administrowanie produktem IBM MQ. Więcej informacji 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 klientów na wielu platformach” na stronie 1177.
2. Aby wywołać jeden z przykładowych programów przy użyciu danych pochodzących z podzbioru PUT w zbiorze AMQSDATA biblioteki QMQMSAMP, należy użyć komendy, takiej jak:

   ```
   CALL PGM(QMQM/AMQSPUT4) PARM('QMQMSAMP/AMQSDATA(PUT)')
   ```

   Uwaga: Aby skompilowany moduł mógł korzystać z systemu plików IFS, należy określić opcję SYSIFCOPT (*IFSIO) w komendzie CRTCMOD, a następnie podać nazwę zbioru, przekazanego jako parametr, w następującym formacie:

   ```
   home/me/myfile
   ```
3. Aby użyć wersji języka COBOL w przykładach Inquire, Set i Echo, należy zmienić definicje procesów przed uruchomieniem tych przykładów.

   W przypadku przykładów Inquire, Set i Echo, przykładowe definicje wyzwalają wersje C tych przykładów. Jeśli wersje COBOL mają być używane, 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, patrz Change MQ Process (CHGMQMPRC)), a następnie można edytować i uruchamiać komendę **AMQSAMP4** z alternatywną definicją.
4. Uruchom programy przykładowe.

   Więcej informacji na temat parametrów, których oczekuje każda z przykładów, można znaleźć w opisach poszczególnych przykładów.

   Uwaga: W przypadku przykładowych programów w języku COBOL, gdy nazwy kolejek są przekazywane jako parametry, należy podać 48 znaków, dopełnianie pustych znaków, jeśli to konieczne. Wszystkie inne znaki niż 48 znaków powodują, że program nie powiedzie się z kodem przyczyny 2085.

Odsyłacze pokrewne

“Przykłady dla produktu IBM i” na stronie 1174
Techniki pokazanego przez programy przykładowe dla IBM MQ w systemach IBM i .

UNIX *Przygotowywanie i uruchamianie przykładowych programów w systemie UNIX and Linux*

Przed uruchomieniem przykładowych programów w systemie UNIXnależy najpierw utworzyć menedżer kolejek, a także utworzyć potrzebne kolejki. Aby uruchomić przykłady w języku COBOL, konieczne może być wykonanie dodatkowych czynności przygotowawcze.

O tym zadaniu

Przykładowe pliki IBM MQ w systemach UNIX and Linux znajdują się w katalogach wymienionych w sekcji Tabela 167 na stronie 1181 , jeśli wartości domyślne zostały użyte podczas instalacji.

Tabela 167. Gdzie znaleźć przykłady dla produktu IBM MQ w systemach UNIX and Linux	
Zawartość	**Katalog**
Pliki źródłowe	*MQ_INSTALLATION_PATH*/samp
pliki źródłowe procedur obsługi kolejki niedostarczonych komunikatów	*MQ_INSTALLATION_PATH*/samp/dlq
Pliki wykonywalne	*MQ_INSTALLATION_PATH*/samp/bin

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

W przypadku przykładów wymagany jest zestaw kolejek do pracy. Aby utworzyć zestaw, można użyć własnych kolejek lub uruchomić przykładowy plik MQSC amqscos0.tst . Aby uruchomić przykłady, należy użyć dostarczonych wersji kodu wykonywalnego lub skompilować wersje źródłowe tak, jak w przypadku innych aplikacji, używając kompilatora ANSI.

Procedura

1. Utwórz menedżer kolejek i skonfiguruj domyślne definicje.

   Należy to zrobić, aby możliwe było uruchomienie dowolnego z programów przykładowych. Więcej informacji na temat tworzenia menedżera kolejek zawiera sekcja Administrowanie produktem IBM MQ. Więcej informacji 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 klientów na wielu platformach” na stronie 1177.
2. Jeśli nie korzystasz z własnych kolejek, uruchom przykładowy plik MQSC amqscos0.tst , aby utworzyć zestaw kolejek.

   Aby wykonać tę czynności w systemach UNIX and Linux , wpisz:

   ```
   runmqsc QManagerName <amqscos0.tst > /tmp/sampobj.out
   ```

   Sprawdź plik `sampobj.out` , aby upewnić się, że nie wystąpiły żadne błędy.
3. Aby użyć wersji języka COBOL w przykładach Inquire, Set i Echo, należy zmienić definicje procesów przed uruchomieniem tych przykładów.

   W przypadku przykładów Inquire, Set i Echo, przykładowe definicje wyzwalają wersje C tych przykładów. Jeśli wersje COBOL mają być używane, należy zmienić definicje procesów:

   - SYSTEM.SAMPLE.INQPROCESS
   - SYSTEM.SAMPLE.SETPROCESS
   - SYSTEM.SAMPLE.ECHOPROCESS

   W systemie Windowsnależy wykonać następujące czynności, edytując plik `amqscos0.tst` i zmieniając nazwy plików wykonywalnych języka C na nazwy plików wykonywalnych w języku COBOL przed użyciem komendy **runmqsc** w celu uruchomienia tych przykładów.
4. Uruchom programy przykładowe.

   Aby uruchomić przykład, wprowadź jego nazwę, a następnie dowolne parametry, na przykład:

   ```
   amqsput myqueue qmanagername
   ```

   gdzie *myqueue* jest nazwą kolejki, w której mają zostać umieszczone komunikaty, a *qmanagername* jest menedżerem kolejek, który jest właścicielem produktu *myqueue*.

   Więcej informacji na temat parametrów, których oczekuje każda z przykładów, można znaleźć w opisach poszczególnych przykładów.

Uwaga: W przypadku przykładowych programów w języku COBOL, gdy nazwy kolejek są przekazywane jako parametry, należy podać 48 znaków, dopełnianie pustych znaków, jeśli to konieczne. Wszystkie inne znaki niż 48 znaków powodują, że program nie powiedzie się z kodem przyczyny 2085.

Odsyłacze pokrewne

"Przykłady dla systemów UNIX and Linux" na stronie 1168
Techniki pokazanego przez programy przykładowe dla IBM MQ w systemach UNIX and Linux .

Windows *Przygotowywanie i uruchamianie przykładowych programów w systemie Windows*

Przed uruchomieniem przykładowych programów w systemie Windowsnależy najpierw utworzyć menedżer kolejek, a także utworzyć potrzebne kolejki. Aby uruchomić przykłady w języku COBOL, konieczne może być wykonanie dodatkowych czynności przygotowawcze.

O tym zadaniu

Przykładowe pliki produktu IBM MQ for Windows znajdują się w katalogach wymienionych w sekcji Tabela 168 na stronie 1182, jeśli wartości domyślne zostały użyte w czasie instalacji. Wartością domyślną napędu instalacyjnego jest < c: >.

Tabela 168. Gdzie znaleźć przykłady dla produktu IBM MQ for Windows

Zawartość	**Katalog**
Kod źródłowy C	*MQ_INSTALLATION_PATH*\Tools\C\Samples
Kod źródłowy dla przykładu procedury 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 (32-bitowe wersje) *MQ_INSTALLATION_PATH*\ Tools\C\Samples\Bin64 (wersje 64-bitowe)
Przykładowe pliki MQSC	*MQ_INSTALLATION_PATH*\Tools\MQSC\Samples
Kod źródłowy Visual Basic	*MQ_INSTALLATION_PATH*\Tools\VB\SampVB6
.NET – przykłady	*MQ_INSTALLATION_PATH*\Tools\dotnet \ Przykłady

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Uwaga: Wersje 64-bitowe są dostępne dla niektórych plików wykonywalnych w języku C.

W przypadku przykładów wymagany jest zestaw kolejek do pracy. Aby utworzyć zestaw kolejek, można użyć własnych kolejek lub uruchomić przykładowy plik MQSC amqscos0.tst . Aby uruchomić przykłady, należy użyć dostarczonych wersji kodu wykonywalnego lub skompilować wersje źródłowe tak, jak w przypadku innych aplikacji produktu IBM MQ for Windows .

Procedura

1. Utwórz menedżer kolejek i skonfiguruj domyślne definicje.

   Należy to zrobić, aby możliwe było uruchomienie dowolnego z programów przykładowych. Więcej informacji na temat tworzenia menedżera kolejek zawiera sekcja Administrowanie produktem IBM MQ. Więcej informacji 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 klientów na wielu platformach" na stronie 1177.

2. Jeśli nie korzystasz z własnych kolejek, uruchom przykładowy plik MQSC amqscos0.tst , aby utworzyć zestaw kolejek.

   Aby wykonać tę komendę 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.

   Uwaga:

3. Aby użyć wersji języka COBOL w przykładach Inquire, Set i Echo, należy zmienić definicje procesów przed uruchomieniem tych przykładów.

   W przypadku przykładów Inquire, Set i Echo, przykładowe definicje wyzwalają wersje C tych przykładów. Jeśli wersje COBOL mają być używane, należy zmienić definicje procesów:

   - SYSTEM.SAMPLE.INQPROCESS
   - SYSTEM.SAMPLE.SETPROCESS
   - SYSTEM.SAMPLE.ECHOPROCESS

   W systemie Windowsnależy wykonać następujące czynności, edytując plik `amqscos0.tst` i zmieniając nazwy plików wykonywalnych języka C na nazwy plików wykonywalnych w języku COBOL przed użyciem komendy **runmqsc** w celu uruchomienia tych przykładów.

4. Uruchom programy przykładowe.

   Aby uruchomić przykład, wprowadź jego nazwę, a następnie dowolne parametry, na przykład:

   ```
   amqsput myqueue qmanagername
   ```

   gdzie *myqueue* jest nazwą kolejki, w której mają zostać umieszczone komunikaty, a *qmanagername* jest menedżerem kolejek, który jest właścicielem produktu *myqueue*.

   Więcej informacji na temat parametrów, których oczekuje każda z przykładów, można znaleźć w opisach poszczególnych przykładów.

   Uwaga: W przypadku przykładowych programów w języku COBOL, gdy nazwy kolejek są przekazywane jako parametry, należy podać 48 znaków, dopełnianie pustych znaków, jeśli to konieczne. Wszystkie inne znaki niż 48 znaków powodują, że program nie powiedzie się z kodem przyczyny 2085.

Odsyłacze pokrewne

“Przykłady dla produktu IBM MQ for Windows” na stronie 1171
Techniki pokazanego przez programy przykładowe dla produktu IBM MQ for Windows.

“Przykłady Visual Basic dla produktu IBM MQ for Windows” na stronie 1174
Techniki pokazanego przez programy przykładowe dla IBM MQ w systemach Windows .

Przykładowy program obsługi wyjścia funkcji API

Przykładowe wyjście interfejsu API generuje dane śledzenia MQI w pliku określonym 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 na wielu platformach” na stronie 1050.

Źródło
amqsaxe0.c

Binarny
amqsaxe

Konfigurowanie dla wyjścia przykładu

1. Dodaj następujące informacje do 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* oznacza 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* oznacza katalog, w którym zainstalowano produkt IBM MQ .

2. Ustaw zmienną środowiskową

```
MQAPI_TRACE_LOGFILE=/tmp/MqiTrace
```

3. Uruchom aplikację.

   Pliki wyjściowe są tworzone w katalogu /tmp z nazwami takimi jak: `MqiTrace.`*`pid`*`.`*`tid`*`.log`

Przykładowy program konsumpcji asynchronicznej

Przykładowy program amqscbf demonstruje użycie komendy MQCB i MQCTL w celu asynchronicznego korzystania z komunikatów z wielu kolejek.

amqscbf jest dostarczany jako kod źródłowy C, a klient binarny i plik wykonywalny serwera na platformach Windows, UNIX and Linux .

Program jest uruchamiany z poziomu 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 odczytać komunikaty z wielu kolejek (maksymalnie dziesięć kolejek jest obsługiwanych przez próbkę).

Uwaga: Reconnect type jest poprawna tylko dla programów klienckich.

Przykład

W przykładzie przedstawiono komendę amqscbf run as a server program reading one message from QL1 and then being zatrzymany.

Użyj programu IBM MQ Explorer, aby umieścić komunikat testowy w produkcie 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 demonstruje amqscbf

W przykładzie pokazano, jak odczytywać komunikaty z wielu kolejek w kolejności ich przybycia. Wymagałoby to dużo większej liczby kodów przy użyciu synchronicznego wywołania MQGET. W przypadku wykorzystania asynchronicznego nie jest wymagane odpytywanie, a zarządzanie wątkami i pamięcią masową jest wykonywane przez produkt IBM MQ. Przykład "realny świat" musiałby poradzić sobie z błędami; w przykładowych błędach wypisuje się na konsolę.

Przykładowy kod składa się z następujących kroków:

1. Zdefiniuj pojedynczą funkcję zwrotną wykorzystania komunikatów,

```
void MessageConsumer(MQHCONN      hConn,
                     MQMD    * pMsgDesc,
                     MQGMO   * pGetMsgOpts,
                     MQBYTE  * Buffer,
                     MQCBC   * pContext)
{ ... }
```

2. Połącz się z menedżerem kolejek,

```
MQCONNX(QMName,&cno,&Hcon,&CompCode,&CReason);
```

3. Otwórz kolejki wejściowe i powiąz każdą z nich za pomocą funkcji zwrotnej `MessageConsumer` ,

```
MQOPEN(Hcon,&od,O_options,&Hobj,&OpenCode,&Reason);
cbd.CallbackFunction = MessageConsumer;
MQCB(Hcon,MQOP_REGISTER,&cbd,Hobj,&md,&gmo,&CompCode,&Reason);
```

   `cbd.CallbackFunction` nie musi być ustawiany dla każdej kolejki; jest to pole tylko wejściowe. Możliwe jest jednak powiązanie innej funkcji zwrotnej z każdą kolejką.

4. Rozpoczęcie korzystania z komunikatów,

```
MQCTL(Hcon,MQOP_START,&ctlo,&CompCode,&Reason);
```

5. Poczekaj, aż użytkownik kliknie klawisz Enter, a następnie zaprzestanie korzystania z komunikatów,

```
MQCTL(Hcon,MQOP_STOP,&ctlo,&CompCode,&Reason);
```

6. W końcu rozłącz się z menedżerem kolejek,

```
MQDISC(&Hcon,&CompCode,&Reason);
```

Przykładowy program do umieszczania w pamięci asynchronicznej

Sekcja zawiera informacje na temat uruchamiania przykładu amqsapt oraz projektu przykładowego programu Asynchroniczne umieszczanie przykładu.

Program przykładowy wstawiany asynchronicznie umieszcza komunikaty w kolejce przy użyciu wywołania asynchronicznego MQPUT, a następnie pobiera informacje o statusie za pomocą wywołania MQSTAT. Nazwę tego programu można znaleźć w sekcji “Funkcje demonstruje w przykładowych programach na Multiplatforms” na stronie 1168 na różnych platformach.

Uruchamianie przykładu amqsapt

Ten program przyjmuje maksymalnie 6 parametrów:

1. Nazwa kolejki docelowej (wymagane)

2. Nazwa menedżera kolejek (opcjonalnie)
3. Opcje otwarcia (opcjonalnie)
4. Opcje zamknięcia (opcjonalnie)
5. Nazwa docelowego menedżera kolejek (opcjonalnie)
6. Nazwa kolejki dynamicznej (opcjonalnie)

Jeśli menedżer kolejek nie jest określony, amqsapt łączy się z domyślnym menedżerem kolejek.

Projekt przykładowego programu Asynchronicznego Put

Program korzysta z wywołania MQOPEN z dostarczonym opcjami wyjścia lub z opcjami MQOO_OUTPUT i MQOO_FAIL_IF_QUIESCING, aby otworzyć kolejkę docelową w celu umieszczania komunikatów.

Jeśli nie jest możliwe otwarcie kolejki, program wyświetli komunikat o błędzie zawierający kod przyczyny zwrócony przez wywołanie MQOPEN. Aby program był prosty, w tym 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 asynchronicznie umieszczonego go w kolejce docelowej. Program jest kontynuowany do momentu osiągnięcia końca danych wejściowych lub wywołanie MQPUT nie powiedzie się. Jeśli program dociera do końca danych wejściowych, zamyka kolejkę za pomocą wywołania MQCLOSE.

Następnie program wysyła wywołanie MQSTAT, zwracając strukturę MQSTS, a następnie wyświetla komunikaty zawierające liczbę pomyślnie umieszczonych komunikatów, liczbę komunikatów wysłanych z ostrzeżeniem oraz liczbę niepowodzeń.

Przeglądanie przykładowych programów

W przykładowych programach przeglądania komunikaty są przeglądane w kolejce przy użyciu wywołania MQGET.

Nazwy tych programów można znaleźć w sekcji “Funkcje demonstruje w przykładowych programach na Multiplatforms” na stronie 1168 .

Projekt przykładowego programu przeglądania

Program otwiera kolejkę docelową przy użyciu wywołania MQOPEN z opcją MQOO_BROWSE. Jeśli nie jest możliwe otwarcie kolejki, program wyświetli komunikat o błędzie zawierający kod przyczyny zwrócony przez wywołanie MQOPEN.

Dla każdego komunikatu w kolejce program korzysta z wywołania MQGET w celu 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 ustawiony logicznie przed pierwszym komunikatem w kolejce, dlatego ta opcja powoduje zwrócenie komunikatu ***pierwszy*** , gdy wywołanie jest wykonywane po raz pierwszy.

MQGMO_NO_WAIT
Program nie czeka, jeśli w kolejce nie ma żadnych komunikatów.

Komunikat MQGMO_ACCEPT_TRUNCATED_MSG
Wywołanie MQGET określa bufor o stałej wielkości. Jeśli komunikat jest dłuższy niż ten bufor, w programie zostanie wyświetlony obcięty komunikat wraz z ostrzeżeniem, że komunikat został obcięty.

Program demonstruje sposób, w jaki należy wyczyścić pola *MsgId* i *CorrelId* struktury MQMD po każdym wywołaniu MQGET, ponieważ wywołanie ustawia te pola na wartości zawarte w komunikacie, który pobiera. Usunięcie zaznaczenia tych pól oznacza, że kolejne wywołania MQGET pobierają komunikaty w kolejności, w jakiej komunikaty są przechowywane w kolejce.

Program będzie kontynuował działanie na końcu kolejki; wywołanie MQGET zwraca kod przyczyny MQRC_NO_MSG_AVAILABLE, a program wyświetli komunikat ostrzegawczy. Jeśli wywołanie MQGET nie powiedzie się, w programie zostanie wyświetlony komunikat o błędzie zawierający kod przyczyny.

Następnie program zamknie kolejkę przy użyciu wywołania MQCLOSE.

Przeglądaj przykładowe programy dla programu UNIX, Linux, and Windows
Warto rozważyć użycie tego tematu podczas poznawania programów przykładowych w programie UNIX, Linux, and Windows.

Wersja C programu przyjmuje 2 parametry

1. Nazwa kolejki źródłowej (niezbędne)
2. Nazwa menedżera kolejek (opcjonalnie)

Jeśli menedżer kolejek nie jest określony, łączy się z domyślnym menedżerem kolejek. Na przykład wprowadź jedną z następujących opcji:

- `amqsgbr myqueue qmanagername`
- `amqsgbrc myqueue qmanagername`
- `amq0gbr0 myqueue`

gdzie `myqueue` to nazwa kolejki, z której będą wyświetlane komunikaty, a `qmanagername` to menedżer kolejek, który jest właścicielem produktu `myqueue`.

If you omit the `qmanagername`, when running the C sample, it assumes that the default queue manager owns the queue.

Wersja COBOL nie ma żadnych parametrów. Jest on połączony z domyślnym menedżerem kolejek i po uruchomieniu zostanie wyświetlony monit:

```
Please enter the name of the target queue
```

Wyświetlane są tylko pierwsze 50 znaków każdego komunikatu, po którym następuje `- - - truncated` , gdy jest to przypadek.

Przeglądanie przykładowych programów w systemie IBM i
Każdy program pobiera kopie wszystkich komunikatów znajdujących się w kolejce, które zostały określone podczas wywoływania programu; komunikaty pozostają w kolejce.

Można użyć podanej kolejki SYSTEM.SAMPLE.LOCAL; najpierw uruchom przykładowy program umieszczania w celu umieszczenia niektórych komunikatów w kolejce. W tym celu można użyć kolejki SYSTEM.SAMPLE.ALIAS, która jest nazwą aliasu dla tej samej kolejki lokalnej. Program będzie kontynuowany do momentu osiągnięcia końca kolejki lub wywołanie MQI nie powiedzie się.

Przykłady C umożliwiają określenie nazwy menedżera kolejek, zwykle jako drugiego parametru, w podobny sposób, jak w przypadku przykładów systemów Windows . Na przykład:

```
CALL PGM(QMQM/AMQSTRG4) PARM('SYSTEM.SAMPLE.TRIGGER' 'QM01')
```

Jeśli menedżer kolejek nie jest określony, łączy się z domyślnym menedżerem kolejek. Jest to również istotne dla przykładów RPG. Jednak w przypadku przykładów w języku RPG należy podać nazwę menedżera kolejek, zamiast zezwalać na jej ustawienie domyślne.

ULW *Przykładowy program przeglądarki*

Przykładowy program przeglądarki odczytuje i zapisuje zarówno deskryptor komunikatu, jak i pola treści komunikatu dla wszystkich komunikatów znajdujących się w kolejce.

Przykładowy program jest napisany jako program narzędziowy, nie tylko po to, aby zademonstrować technikę. Nazwy tych programów można znaleźć w sekcji “Funkcje demonstruje w przykładowych programach na Multiplatforms” na stronie 1168 .

Ten program przyjmuje następujące parametry pozycyjne:

1. Nazwa kolejki źródłowej (wymagane)
2. Nazwa menedżera kolejek (wymagane)
3. Opcjonalny parametr właściwości (opcjonalny)

Programy te korzystają również ze zmiennej środowiskowej o nazwie **MQSAMP_USER_ID** , która powinna być ustawiona na identyfikator użytkownika, który ma być używany do uwierzytelniania połączenia. Po ustawieniu tej opcji program prosi o podanie hasła, które ma towarzyszyć temu identyfikatorowi użytkownika.

Aby uruchomić te programy, wprowadź jedną z następujących komend:

- `amqsbcg` *`myqueue qmanagername`*
- `amqsbcgc` *`myqueue qmanagername`*

gdzie *myqueue* to nazwa kolejki, w której mają być przeglądane komunikaty, a *qmanagername* to menedżer kolejek, który jest właścicielem produktu *myqueue*.

Odczytuje on każdy komunikat z kolejki i zapisuje następujące informacje w celu wyjścia standardowego wyjścia:

- Sformatowane pola deskryptora komunikatu
- Dane komunikatu (zrzucone w formacie szesnastkowym i, gdzie jest to możliwe, format znakowy)

Tabela 169. Dopuszczalne wartości parametru właściwości

Wartość	**Zachowanie**
0	Domyślne zachowanie, tak jak w przypadku produktu IBM WebSphere MQ 6. Właściwości, które są dostarczane do aplikacji, są zależne od atrybutu kolejki **PropertyControl** , z którego pochodzi komunikat.

1	Uchwyt komunikatu jest tworzony i używany razem z MQGET. Właściwości komunikatu, z wyjątkiem tych, które znajdują się w deskryptorze komunikatu (lub rozszerzeniu), są wyświetlane w podobny sposób do deskryptora komunikatu. Na przykład:
	`****Message properties****`
	`  property name: property value`
	Lub jeśli żadne właściwości nie są dostępne:
	`****Message properties****`
	`  None`
	Wartości liczbowe są wyświetlane przy użyciu `printf`, wartości łańcuchowe są otaczane w pojedynczych cudzysłowach, a łańcuchy bajtowe są otoczone znakiem X i apostrofami, co dla deskryptora komunikatu.

2	Określono parametr MQGMO_NO_PROPERTIES, tak aby zwracane były tylko właściwości deskryptora komunikatu.
3	Określono wartość MQGMO_PROPERTIES_FORCE_MQRFH2 , tak aby wszystkie właściwości zostały zwrócone w danych komunikatu.
4	Właściwość MQGMO_PROPERTIES_COMPATIBILITY jest określona, tak aby wszystkie właściwości mogły zostać zwrócone w zależności od tego, czy właściwość IBM WebSphere MQ 6 jest włączona, w przeciwnym razie właściwości są usuwane.

Program jest ograniczony do drukowania pierwszych 65535 znaków wiadomości i kończy się niepowodzeniem z powodu `truncated msg` , jeśli odczytany zostanie dłuższy komunikat.

Przykład danych wyjściowych z tego programu narzędziowego znajduje się w sekcji Przeglądanie kolejek.

Przykład transakcji CICS

Dostępny jest przykładowy program transakcyjny CICS o nazwie amqscic0.ccs dla kodu źródłowego i amqscic0 dla wersji wykonywalnej. Transakcje można tworzyć za pomocą standardowych narzędzi CICS .

Szczegółowe informacje na temat komend wymaganych dla danej platformy zawiera sekcja “Budowanie aplikacji proceduralnej” na stronie 1099 .

Transakcja odczytuje komunikaty z kolejki transmisji SYSTEM.SAMPLE.CICS.WORKQUEUE w domyślnym menedżerze kolejek i umieszcza je w kolejce lokalnej, której nazwa znajduje się w nagłówku transmisji komunikatu. Wszystkie niepowodzenia są wysyłane do kolejki SYSTEM.SAMPLE.CICS.DLQ.

Uwaga: Aby utworzyć te kolejki i przykładowe kolejki wejściowe, można użyć przykładowego skryptu MQSC amqscic0.tst .

Przykładowy program Connect

Przykładowy program Connect umożliwia eksplorowanie wywołania MQCONNX i jego opcji z poziomu klienta. Ten przykład łączy się z menedżerem kolejek przy użyciu wywołania MQCONNX, zapytanie o nazwę menedżera kolejek za pomocą wywołania MQINQ i wyświetlenie go. Dowiedz się również o uruchomieniu przykładu amqscnxc.

Uwaga: Przykładowy program Connect jest przykładowym klientem. Można go skompilować i uruchomić na serwerze, ale funkcja ma znaczenie tylko na kliencie, a tylko pliki wykonywalne klienta są dostarczane.

Uruchamianie przykładu amqscnxc

Składnia wiersza komend programu 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, o ile został określony, musi być ostatni. Parametry są następujące:

ConnName
Nazwa połączenia TCP/IP menedżera kolejek serwera

Jeśli nazwa połączenia TCP/IP nie zostanie określona, dla parametru MQCONNX zostanie wystawiona wartość *ClientConnPtr* , a wartość NULL jest ustawiona na NULL.

Nazwa SvrconnChannel
Nazwa kanału połączenia z serwerem

Jeśli zostanie określona nazwa połączenia TCP/IP, ale nie jest to kanał połączenia z serwerem (odwrócenie nie jest dozwolone), to przykład użyje nazwy SYSTEM.DEF.SVRCONN.

Użytkownik
Nazwa użytkownika, która ma być używana na potrzeby uwierzytelniania połączenia

Jeśli zostanie podany, program wyświetli monit o podanie hasła, które będzie towarzyszyć temu identyfikatorowi użytkownika.

QMgrName
Nazwa docelowego menedżera kolejek

Jeśli docelowy menedżer kolejek nie zostanie określony, próbka łączy się z tym, który menedżer kolejek będzie nasłuchiwać przy danej nazwie połączenia TCP/IP.

Uwaga: Jeśli jako jedyny parametr zostanie wprowadzony znak zapytania lub jeśli zostaną wprowadzone niepoprawne parametry, zostanie wyświetlony komunikat z wyjaśnieniem sposobu korzystania z programu.

Jeśli próbka zostanie uruchomiona bez opcji wiersza komend, do określenia informacji o połączeniu zostanie użyta zawartość zmiennej środowiskowej MQSERVER. (W tym przykładzie wartość MQSERVER

jest ustawiona na SYSTEM.DEF.SVRCONN/TCP/machine.site.company.com.) Zostaną wyświetlone dane wyjściowe podobne do następujących:

```
Sample AMQSCNXC start
Connecting to the default queue manager
with no client connection information specified.
Connection established to queue manager machine

Sample AMQSCNXC end
```

Jeśli uruchamiasz przykład i podaj nazwę połączenia TCP/IP i nazwę kanału połączenia z serwerem, ale nie ma nazwy docelowego menedżera kolejek, tak jak to:

```
amqscnxc -x machine.site.company.com -c SYSTEM.ADMIN.SVRCONN
```

Domyślna nazwa menedżera kolejek jest używana, a dane wyjściowe są podobne do następujących:

```
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
```

W przypadku uruchomienia przykładu podaj nazwę połączenia TCP/IP i nazwę docelowego menedżera kolejek, tak jak to:

```
amqscnxc -x machine.site.company.com MACHINE
```

Wyświetlane są 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 konwersji danych jest szkieletem procedury wyjścia konwersji danych. Informacje na temat projektowania przykładu konwersji danych.

Nazwy tych programów można znaleźć w sekcji “Funkcje demonstruje w przykładowych programach na Multiplatforms” na stronie 1168 .

Projekt próby konwersji danych

Każda procedura wyjścia konwersji danych przekształca pojedynczy nazwany format komunikatu. Ten szkielet jest przeznaczony jako opakowanie dla fragmentów kodu generowanych przez program narzędziowy do generowania danych wyjściowych konwersji danych.

Program narzędziowy tworzy jeden fragment kodu dla każdej struktury danych; kilka takich struktur tworzy format, więc do tego szkieletu dodawane są kilka fragmentów kodu w celu utworzenia procedury do konwersji danych w całym formacie.

Następnie program sprawdza, czy konwersja jest sukcesem lub niepowodzeniem, a następnie zwraca wartości wymagane do programu wywołującego.

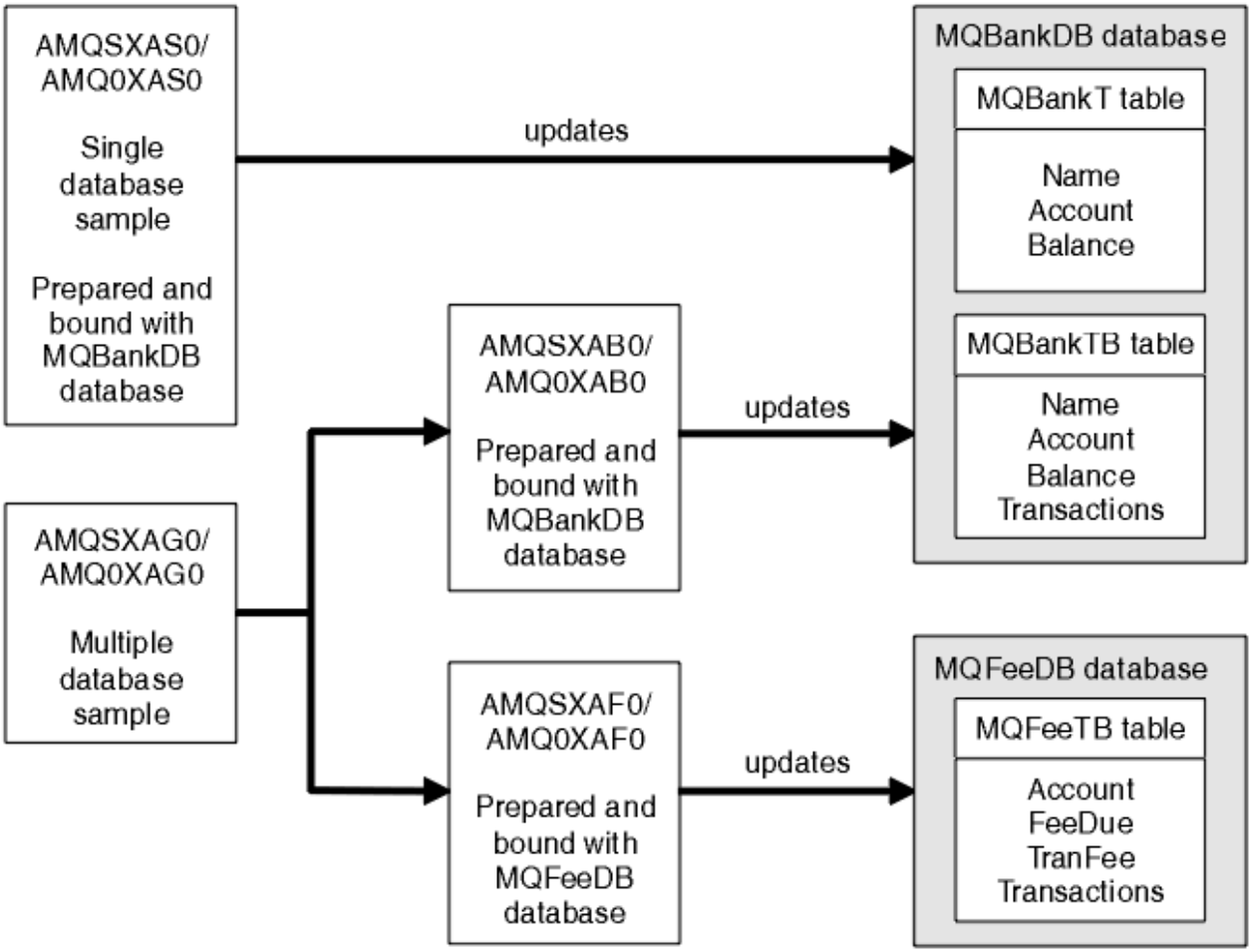
Przykłady koordynacji bazy danych

Udostępniono dwie próbki, które demonstrują, w jaki sposób produkt IBM MQ może koordynować zarówno aktualizacje produktu IBM MQ , jak i aktualizacje bazy danych w ramach tej samej jednostki pracy.

Są to następujące przykłady:

1. AMQSXAS0 (w języku C) lub AMQ0XAS0 (w języku COBOL), które aktualizuje pojedynczą bazę danych w ramach 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 wspólnie aktualizują dwie bazy danych w ramach jednostki pracy produktu IBM MQ , pokazując, w jaki sposób można uzyskać dostęp do wielu baz danych. Te przykłady są udostępniane w celu wyświetlenia użycia wywołań MQBEGIN, mieszanych wywołań SQL i IBM MQ , a także w przypadku, gdy i kiedy ma zostać nawiązane połączenie z bazą danych.

Rysunek 133 na stronie 1191 pokazuje, w jaki sposób udostępnione przykłady służą do aktualizowania baz danych:



Rysunek 133. Przykłady koordynacji bazy danych

Programy odczytają komunikat z kolejki (w punkcie synchronizacji), a następnie, korzystając z informacji znajdujących się w komunikacie, uzyskają odpowiednie informacje z bazy danych i aktualizują je. Następnie zostanie wydrukowany nowy status bazy danych.

Logika programu jest następująca:

1. Użyj nazwy kolejki wejściowej z argumentu programu

2. Nawiąże połączenie z domyślnym menedżerem kolejek (lub opcjonalnie dostarczonym nazwą w języku C) przy użyciu komendy MQCONN.
3. Otwórz kolejkę (za pomocą komendy MQOPEN) dla danych wejściowych, gdy nie występują niepowodzenia.
4. Uruchamianie jednostki pracy przy użyciu komendy MQBEGIN
5. Pobierz następny komunikat (za pomocą komendy MQGET) z kolejki w punkcie synchronizacji
6. Pobierz informacje z baz danych
7. Aktualizacja informacji z baz danych
8. Zatwierdź zmiany za pomocą MQCMIT
9. Wydrukuj zaktualizowane informacje (brak dostępnych komunikatów zliczanych jako niepowodzenie, a pętla kończy się)
10. Zamknij kolejkę za pomocą komendy MQCLOSE
11. Rozłącz się z kolejką za pomocą MQDISC

W próbkach używane są kursory SQL, dzięki czemu odczyty z baz danych (czyli wiele instancji) są blokowane w czasie przetwarzania komunikatu, co pozwala na jednoczesne uruchamianie wielu instancji tych programów. Kursory są jawnie otwierane, ale niejawnie zamykane przy użyciu wywołania MQCMIT.

Pojedyncza próbka bazy danych (AMQSXAS0 lub AMQ0XAS0) nie zawiera instrukcji SQL CONNECT, a połączenie z bazą danych jest niejawnie wykonywane przez produkt IBM MQ przy użyciu wywołania MQBEGIN. Przykładowa baza danych (AMQSXAG0 lub AMQ0XAG0, AMQSXAB0 lub AMQ0XAB0oraz AMQSXAF0 lub AMQ0XAF0) zawiera instrukcje SQL CONNECT, ponieważ niektóre produkty bazodanowe zezwalają na tylko jedno aktywne połączenie. Jeśli nie jest to przypadek dla produktu bazodanowego lub jeśli użytkownik uzyskuje dostęp do pojedynczej bazy danych w wielu produktach bazodanowych, instrukcje SQL CONNECT mogą zostać usunięte.

Przykłady są przygotowywane razem z produktem bazodanowym IBM Db2 , dlatego może być konieczne zmodyfikowanie ich w celu pracy z innymi produktami bazodanowymi.

Sprawdzanie błędów SQL używa podprogramów w UTIL.C i CHECKERR.CBL dostarczane przez produkt Db2. Muszą one zostać skompilowane lub zastąpione przed kompilacją i dowiązowaniem.

Uwaga: Jeśli używane jest źródło Micro Focus COBOL, CHECKERR.MFC dla sprawdzania błędów SQL, należy zmienić ID programu na wielkie, czyli CHECKERR, dla AMQ0XAS0 , aby poprawnie dowiązali się.

Tworzenie baz danych i tabel
Utwórz bazy danych i tabele przed kompilacją przykładów.

Aby utworzyć bazy danych, należy użyć zwykłej metody dla produktu bazy danych, na przykład:

```
DB2 CREATE DB MQBankDB
DB2 CREATE DB MQFeeDB
```

Utwórz tabele przy użyciu 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 przy użyciu 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 używać tych samych instrukcji SQL, ale END_EXEC na końcu każdego wiersza.

Prekompilowanie, kompilowanie i łączenie próbek
Ta sekcja zawiera informacje na temat prekompilacji, kompilowania i łączenia przykładów w językach C i COBOL.

Prekompiluj pliki `.SQC` (w języku C) i pliki `.SQB` (w języku COBOL) i powiąż je z odpowiednią bazą danych w celu utworzenia plików `.C` lub `.CBL` . W tym celu należy użyć typowej metody dla produktu bazodanowego.

Wstępne kompilowanie 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
```

Prekompilowanie 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

Następujące przykładowe komendy używają symboli *DB2TOP* i *MQ_INSTALLATION_PATH*. *DB2TOP* reprezentuje katalog instalacyjny dla produktu Db2 . *MQ_INSTALLATION_PATH* reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

- AIX W systemie AIXścieżka do katalogu jest następująca:

  ```
  /usr/lpp/db2_05_00
  ```

- Solaris W systemach Solarisścieżka do katalogu jest następująca:

  ```
  /opt/IBMdb2/V5.0
  ```

- Windows W systemach Windows ścieżka katalogu zależy od ścieżki wybranej podczas instalowania produktu. Jeśli wybrano ustawienia domyślne, ścieżka jest następująca:

  ```
  c:\sqllib
  ```

Uwaga: Przed wprowadzeniem komendy łącza w systemach Windows należy upewnić się, że zmienna środowiskowa LIB zawiera ścieżki do bibliotek produktu Db2 i IBM MQ .

Skopiuj następujące pliki do katalogu tymczasowego:

- Plik `amqsxag0.c` z instalacji produktu IBM MQ

  Uwaga: Plik ten można znaleźć w następujących katalogach:

  – Linux UNIX W systemach UNIX and Linux:

  ```
  MQ_INSTALLATION_PATH/samp/xatm
  ```

  – Windows W systemach Windows:

  ```
  MQ_INSTALLATION_PATH\tools\c\samples\xatm
  ```

- Pliki `.c` , które zostały uzyskane przez wstępne kompilowanie plików źródłowych `.sqc` , `amqsxas0.sqc`, `amqsxaf0.sqc`i `amqsxab0.sqc`.
- Pliki `util.c` i `util.h` z instalacji produktu Db2 .

  Uwaga: Pliki te można znaleźć w katalogu:

```
DB2TOP/samples/c
```

Zbuduj pliki obiektów dla każdego pliku `.c` przy użyciu następującej komendy kompilatora dla używanej platformy:

- AIX AIX

```
xlc_r -I MQ_INSTALLATION_PATH/inc -I DB2TOP/include -c -o
FILENAME.o FILENAME.c
```

- Solaris Solaris

```
cc -Aa -KPIC -mt -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
```

Utwórz plik wykonywalny `amqsxag0` za pomocą następującej komendy łącza 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
```

- Solaris Solaris

```
cc -mt -L DB2TOP/lib -ldb2 -L MQ_INSTALLATION_PATH/lib
-lmqm -lthread -lsocket -lc -lnsl -ldl 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
```

Utwórz plik wykonywalny produktu `amqsxas0` przy użyciu następujących komend kompilowania i dowiązywania 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
```

- Solaris Solaris

```
cc -mt -L DB2TOP/lib -ldb2-L MQ_INSTALLATION_PATH/lib
-lqm -lthread -lsocket -lc -lnsl -ldl 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 użytkownik pracuje w systemie AIX i chce uzyskać dostęp do bazy danych Oracle, należy użyć kompilatora xlc_r i utworzyć odsyłacz do pliku libmqm_r.a.

Uruchamianie przykładów
Informacje zawarte w tej sekcji 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 użyciem używanego produktu bazodanowego. Informacje na temat sposobu wykonywania tej czynności zawiera sekcja Scenariusz 1: Menedżer kolejek wykonuje koordynację.

Poniższe tytuły zawierają informacje na temat sposobu uruchamiania przykładów w językach C i COBOL:

- “Próbki C” na stronie 1196
- “Przykłady języka COBOL” na stronie 1197

Próbki C

Komunikaty muszą być w następującym formacie, aby można było odczytać z kolejki:

```
UPDATE Balance change=nnn WHERE Account=nnn
```

Komenda AMQSPUT może być używana do umieszczania komunikatów w kolejce.

Przykłady koordynacji bazy danych przyjmują dwa parametry:

1. Nazwa kolejki (wymagane)
2. Nazwa menedżera kolejek (opcjonalnie)

Zakładając, że utworzono i skonfigurowano menedżer kolejek dla pojedynczej próbki bazy danych o nazwie singDBQM, z kolejką o nazwie singDBQ, przyrost konta Freda Bloggsa o 50 jest zwiększany w następujący sposób:

```
AMQSPUT singDBQ singDBQM
```

Następnie klucz w następującym komunikacie:

```
UPDATE Balance change=50 WHERE Account=1
```

W kolejce można umieścić wiele komunikatów.

```
AMQSXAS0 singDBQ singDBQM
```

Następnie zostanie wydrukowany zaktualizowany status konta pana Freda Bloggsa.

Zakładając, że menedżer kolejek został utworzony i skonfigurowany dla przykładu z wieloma bazami danych o nazwie multDBQM, w kolejce o nazwie multDBQ, należy zmniejszyć konto Mary Brown o 75 w następujący sposób:

```
AMQSPUT multDBQ multDBQM
```

Następnie klucz w następującym komunikacie:

```
UPDATE Balance change=-75 WHERE Account=3
```

W kolejce można umieścić wiele komunikatów.

```
AMQSXAG0 multDBQ multDBQM
```

Następnie zostanie wydrukowany zaktualizowany status konta Mary Brown.

Przykłady języka COBOL

Komunikaty muszą być w następującym formacie, aby można było odczytać z kolejki:

```
UPDATE Balance change=snnnnnnnn WHERE Account=nnnnnnnn
```

W przypadku prostoty `Balance change` musi być liczbą 8-znakową podpisaną, a `Account` musi być ośmioznakowym numerem.

W celu umieszczenia komunikatów w kolejce można użyć przykładowego programu AMQSPUT.

Przykłady nie przyjmują żadnych parametrów i korzystają z domyślnego menedżera kolejek. Można go skonfigurować w taki sposób, aby uruchamiał tylko jedną z przykładów w dowolnym momencie. Zakładając, że domyślny menedżer kolejek został skonfigurowany dla pojedynczej próbki bazy danych, z kolejką o nazwie singDBQ, należy zwiększyć konto Freda Bloggsa o 50, wykonując następujące czynności:

```
AMQSPUT singDBQ
```

Następnie klucz w następującym komunikacie:

```
UPDATE Balance change=+00000050 WHERE Account=00000001
```

W kolejce można umieścić wiele komunikatów:

```
AMQ0XAS0
```

Wpisz nazwę kolejki:

```
singDBQ
```

Następnie zostanie wydrukowany zaktualizowany status konta pana Freda Bloggsa.

Zakładając, że domyślny menedżer kolejek został skonfigurowany dla wielokrotnej bazy danych, w kolejce o nazwie multDBQ, należy zmniejszyć konto Mary Brown o 75 w następujący sposób:

```
AMQSPUT multDBQ
```

Następnie klucz w następującym komunikacie:

```
UPDATE Balance change=-00000075 WHERE Account=00000003
```

W kolejce można umieścić wiele komunikatów:

```
AMQ0XAG0
```

Wpisz nazwę kolejki:

```
multDBQ
```

Następnie zostanie wydrukowany zaktualizowany status konta Mary Brown.

Przykład procedury obsługi kolejki niedostarczanych komunikatów

Podana jest przykładowa procedura obsługi kolejki niedostarczonych komunikatów, która jest nazwą pliku wykonywalnego amqsdlq. Jeśli chcesz, aby procedura obsługi kolejki niedostarczonych komunikatów różniła się od RUNMQDLQ, źródło próbki jest dostępne do użycia jako podstawa.

Przykład jest podobny do procedury obsługi niedostarczonych komunikatów udostępnianych w produkcie, ale śledzenie i raportowanie błędów są różne. Istnieją dwie zmienne środowiskowe dostępne dla użytkownika:

ODQ_TRACE
Ustaw wartość YES lub yes, aby włączyć śledzenie

ODQ_MSG
Służy do ustawiania nazwy pliku zawierającego komunikaty o błędach i komunikaty informacyjne. Podany plik nosi nazwę amqsdlq.msg.

Zmienne te muszą być znane w środowisku za pomocą komend **export** lub **set** , w zależności od platformy. Śledzenie jest wyłączone za pomocą komendy **unset** .

Użytkownik może zmodyfikować plik komunikatów o błędach amqsdlq.msg, aby dostosować go do własnych wymagań. Przykład przedstawia komunikaty do wyjścia standardowego, **nie** do pliku dziennika błędów programu IBM MQ .

Podręcznik Administrowanie lub *System Management Guide* dla używanej platformy wyjaśnia, w jaki sposób działa program obsługi niedostarczonych komunikatów i jak go uruchomić.

Przykładowy program listy dystrybucyjnej

Przykład listy dystrybucyjnej amqsptl0 zawiera przykład umieszczania komunikatu w kilku kolejkach komunikatów. Jest on oparty na przykładzie MQPUT amqsput0.

Uruchamianie przykładu Lista dystrybucyjna, amqsptl0

Przykład listy dystrybucyjnej jest uruchamiany w podobny sposób do przykładów umieszczania.

Przyjmuje ona 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 żadna z nazw kolejek lub menedżerów kolejek nie została rozpoznana.

Należy pamiętać o definiowaniu kanałów między menedżerami kolejek, tak aby komunikaty mogły przepływać między nimi. Przykładowy program nie robi tego dla Ciebie.

Projekt przykładu listy dystrybucyjnej

Rekordy komunikatów (put Message Records-MQPMRs) określają atrybuty komunikatów dla każdego miejsca docelowego. Przykład udostępnia wartości dla produktów *MsgId* i *CorrelId*, a te nadpisują wartości określone w strukturze MQMD.

Pole *PutMsgRecFields* w strukturze MQPMO wskazuje, które pola są obecne w strukturze MQPMR:

```
MQLONG PutMsgRecFields=MQPMRF_MSG_ID + MQPMRF_CORREL_ID;
```

Następnie próbka przydziela rekordy odpowiedzi i rekordy obiektów. Rekordy obiektów (MQORs) wymagają co najmniej jednej pary nazw i parzystej liczby nazw, tj. *ObjectName* i *ObjectQMgrName*.

Następny etap obejmuje łączenie się z menedżerami kolejek za pomocą MQCONN. Próbka próbuje połączyć się z menedżerem kolejek powiązanym z pierwszą kolejką w tabeli MQOR. Jeśli ta próba nie powiedzie się, przechodzi ona przez rekordy obiektów z kolei. Użytkownik jest informowany, jeśli nie jest możliwe nawiązanie połączenia z dowolnym menedżerem kolejek i wyjście z programu.

Kolejki docelowe są otwierane za pomocą komendy MQOPEN, a komunikat jest umieszczany w tych kolejkach za pomocą komendy MQPUT. Wszelkie problemy i niepowodzenia są zgłaszane w rekordach odpowiedzi (MQRRs).

W końcu kolejki docelowe są zamykane za pomocą komendy MQCLOSE, a program rozłącza się z menedżerem kolejek za pomocą MQDISC. Te same rekordy odpowiedzi są używane dla każdego wywołania, w którym znajdują się *CompCode* i *Reason*.

Programy przykładowe Echo

Przykładowe programy Echo echo to komunikat z kolejki komunikatów do kolejki odpowiedzi.

Nazwy tych programów można znaleźć w sekcji “Funkcje demonstruje w przykładowych programach na Multiplatforms” na stronie 1168 .

Programy te są przeznaczone do uruchamiania jako programy wyzwalane.

W systemach IBM i UNIX, Linux, and Windows ich jedynym wejściem jest struktura MQTMC2 (komunikat wyzwalacza), która zawiera nazwę kolejki docelowej i menedżera kolejek. W wersji COBOL używany jest domyślny menedżer kolejek.

IBM i W systemie IBM i, aby proces wyzwalający działał, należy upewnić się, że przykładowy program Echo, który ma być używany, jest wyzwalany przez komunikaty przychodzące do kolejki SYSTEM.SAMPLE.ECHO. W tym celu należy podać 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 na ten temat zawiera sekcja Change MQ Process (CHGMQMPRC)). Kolejka przykładowa ma typ wyzwalacza FIRST, dlatego jeśli przed uruchomieniem przykładu żądania w kolejce znajdują się już komunikaty, to próba Echo nie jest wyzwalana przez wysyłane komunikaty.

Jeśli definicja została ustawiona poprawnie, najpierw uruchom program AMQSERV4 w jednym zadaniu, a następnie uruchom komendę AMQSREQ4 w innym. Można użyć komendy AMQSTRG4 zamiast AMQSERV4, ale ewentualne opóźnienia w składaniu zadań mogą spowodować, że śledzenie zdarzeń będzie mniej proste.

Aby wysłać komunikaty do kolejki SYSTEM.SAMPLE.ECHO. Przykładowe programy Echo wysyłają komunikat odpowiedzi zawierający dane zawarte w komunikacie żądania do kolejki odpowiedzi określonej w komunikacie żądania.

Projektowanie przykładowych programów Echo

Program otwiera kolejkę nazwaną w strukturze komunikatu wyzwalacza, która została przekazana podczas jej uruchamiania. (Dla jasności, jest to nazywane kolejką żądań.) Program korzysta z wywołania MQOPEN, aby otworzyć tę kolejkę dla współużytkowanych danych wejściowych.

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 interwałem oczekiwania na 5 sekund. Program testuje deskryptor każdego komunikatu, aby sprawdzić, czy jest to komunikat żądania. Jeśli tak nie jest, program usuwa komunikat i wyświetla komunikat ostrzegawczy.

Dla każdego wiersza danych wejściowych program odczytuje tekst do buforu i korzysta z 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 pole *Feedback* w deskryptorze komunikatu na kod przyczyny zwrócony przez komendę MQGET.

Jeśli w kolejce żądań nie pozostały żadne komunikaty, program zamknie tę kolejkę i rozłącza się z menedżerem kolejek.

IBM i W systemie IBM iprogram może również odpowiadać na komunikaty wysłane do kolejki z platform innych niż IBM MQ for IBM i, chociaż dla tej sytuacji nie jest dostarczana żadna próba. Aby wykonać pracę programu ECHO:

- Napisz program, poprawnie określając parametry **Format**, **Encoding**i **CCSID** , aby wysłać komunikaty żądania tekstowego.

  Program ECHO żąda, aby menedżer kolejek wykonał konwersję danych komunikatów, jeśli jest to potrzebne.
- Określ parametr CONVERT (*YES) w kanale wysyłającym IBM MQ for IBM i , jeśli napisany przez użytkownika program nie zapewnia podobnej konwersji dla odpowiedzi.

Pobieranie przykładowych programów

Program pobierania próbek pobiera komunikaty z kolejki za pomocą wywołania MQGET.

Nazwy tych programów można znaleźć w sekcji “Funkcje demonstruje w przykładowych programach na Multiplatforms” na stronie 1168 .

Projekt przykładowego programu Get

Program otwiera kolejkę docelową przy użyciu wywołania MQOPEN z opcją MQOO_INPUT_AS_Q_DEF. Jeśli nie można otworzyć kolejki, w programie wyświetlany jest komunikat o błędzie zawierający kod przyczyny zwrócony przez wywołanie MQOPEN.

Dla każdego komunikatu w kolejce program korzysta z wywołania MQGET w celu usunięcia komunikatu z kolejki, a następnie wyświetla dane zawarte w komunikacie. Wywołanie MQGET używa opcji MQGMO_WAIT, określając wartość *WaitInterval* 15 sekund, co powoduje, że program oczekuje na ten okres, jeśli w kolejce nie ma komunikatu. Jeśli przed upływem tego okresu nie zostanie wyświetlony żaden komunikat, wywołanie nie powiedzie się i zostanie zwrócony kod przyczyny MQRC_NO_MSG_AVAILABLE.

Program demonstruje sposób, w jaki należy usunąć pola *MsgId* i *CorrelId* struktury MQMD po każdym wywołaniu MQGET, ponieważ wywołanie ustawia te pola na wartości zawarte w komunikacie, który pobiera. Usunięcie zaznaczenia tych pól oznacza, że kolejne wywołania MQGET pobierają komunikaty w kolejności, w jakiej komunikaty są przechowywane w kolejce.

Wywołanie MQGET określa bufor o stałej wielkości. Jeśli komunikat jest dłuższy niż ten bufor, wywołanie nie powiedzie się, a program zostanie zatrzymany.

Program będzie kontynuowany do czasu, aż wywołanie MQGET zwróci kod przyczyny MQRC_NO_MSG_AVAILABLE lub wywołanie MQGET nie powiedzie się. Jeśli wywołanie nie powiedzie się, w programie zostanie wyświetlony komunikat o błędzie zawierający kod przyczyny.

Następnie program zamknie kolejkę przy użyciu 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 nie określono menedżera kolejek, komenda amqsget nawiązuje połączenie z domyślnym menedżerem kolejek, a komenda amqsgetc nawiązuje połączenie z menedżerem kolejek identyfikowanym przez zmienną środowiskową 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.

Programy te używają również zmiennej środowiskowej o nazwie **MQSAMP_USER_ID** , która powinna być ustawiona na identyfikator użytkownika używany do uwierzytelniania połączenia. Jeśli ta opcja jest ustawiona, program poprosi o podanie hasła, które ma towarzyszyć temu identyfikatorowi użytkownika.

Aby uruchomić te programy, wprowadź jedną z następujących komend:

- `amqsget myqueue qmanagername`
- `amqsgetc myqueue qmanagername`

gdzie `myqueue` jest nazwą kolejki, z której program będzie otrzymywał komunikaty, a `qmanagername` jest menedżerem kolejek, do którego należy `myqueue`.

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 **amqsgetc** przy użyciu zmiennych środowiskowych. Na przykład patrz sekcja MQCCDTURL.

Przykład użycia usługi MQSERVER łączącej 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 1204).

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

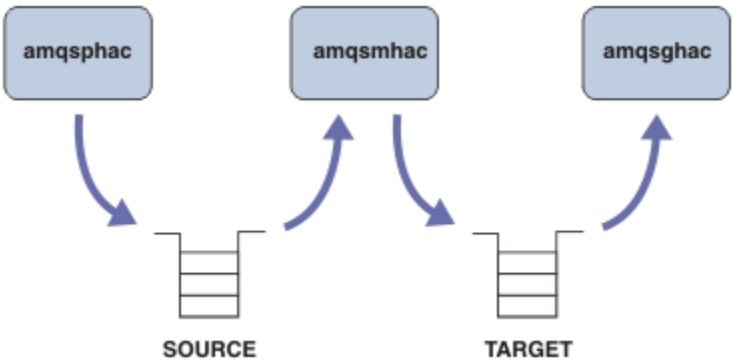
Aby uruchomić przykłady, należy skonfigurować kanał połączenia klienta i serwera. Procedura weryfikacji klienta wyjaśnia sposób konfigurowania środowiska testowego klienta.

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 134 na stronie 1203.



Rysunek 134. 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 1204 w sekcji “Konfigurowanie menedżera kolejek i sterowanie nim” na stronie 1204).
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 1204i 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 Inquire

Przykładowe programy Inquire pytają o niektóre atrybuty kolejki przy użyciu wywołania MQINQ.

Nazwy tych programów można znaleźć w sekcji “Funkcje demonstruje w przykładowych programach na Multiplatforms” na stronie 1168 .

Programy te są przeznaczone do uruchamiania jako programy wyzwalane, więc ich jedynym wejściem jest struktura MQTMC2 (komunikat wyzwalacza) dla systemów IBM i, Windowsi UNIX and Linux . Ta struktura zawiera nazwę kolejki docelowej z atrybutami, które mają zostać zapytane. W wersji C używana jest również nazwa menedżera kolejek. W wersji COBOL używany jest domyślny menedżer kolejek.

Aby proces wyzwalający działał, należy upewnić się, że przykładowy program Inquire, który ma być używany, jest wyzwalany przez komunikaty przychodzące do kolejki SYSTEM.SAMPLE.INQ. W tym celu należy podać nazwę przykładowego programu Inquire, który ma być używany w polu *ApplicId* definicji procesu SYSTEM.SAMPLE.INQPROCESS. IBM i W przypadku produktu IBM imożna użyć komendy CHGMQMPRC, aby uzyskać szczegółowe informacje na ten temat. Więcej informacji na ten temat zawiera sekcja Change MQ Process (CHGMQMPRC). Kolejka przykładowa ma typ wyzwalacza FIRST; jeśli przed uruchomieniem przykładu żądania znajdują się już komunikaty w kolejce, próbka zapytania nie jest wyzwalana przez wysyłane komunikaty.

Jeśli definicja została ustawiona poprawnie:

- ULW W przypadku produktu UNIX, Linux, and Windowsuruchom program **runmqtrm** w jednej sesji, a następnie uruchom program `amqsreq` w innej sesji.
- IBM i W przypadku produktu IBM iuruchom program AMQSERV4 w jednej sesji, a następnie uruchom program AMQSREQ4 w innej sesji. Można użyć komendy AMQSTRG4 zamiast AMQSERV4, ale ewentualne opóźnienia w składaniu zadań mogą spowodować, że śledzenie zdarzeń będzie mniej proste.

Za pomocą przykładowych programów żądania można wysyłać komunikaty żądań, z których każdy zawiera tylko nazwę kolejki, do kolejki SYSTEM.SAMPLE.INQ. W przypadku 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 element pliku wejściowego QMQMSAMP.AMQSDATA(INQ) jest używana, ostatnia kolejka o nazwie nie istnieje, więc próbka zwraca komunikat raportu z kodem przyczyny niepowodzenia.

Projekt przykładowego programu Inquire

Program otwiera kolejkę nazwaną w strukturze komunikatu wyzwalacza, która została przekazana podczas jej uruchamiania. (Dla jasności zadzwonimy do tej *kolejki żądań*.) Program korzysta z wywołania MQOPEN, aby otworzyć tę kolejkę dla współużytkowanych danych wejściowych.

Program korzysta z wywołania MQGET w celu usunięcia komunikatów z tej kolejki. To wywołanie używa opcji MQGMO_ACCEPT_TRUNCATED_MSG i MQGMO_WAIT, z interwałem oczekiwania 5 sekund. Program testuje deskryptor każdego komunikatu, aby sprawdzić, czy jest to komunikat żądania. Jeśli tak nie jest, program usuwa komunikat i wyświetla komunikat ostrzegawczy.

Dla każdego komunikatu żądania usuniętego z kolejki żądań program odczytuje nazwę kolejki (którą wywołamy *kolejkę docelową*). zawarte w danych i otwiera tę kolejkę przy użyciu wywołania MQOPEN z opcją MQOO_INQ. Następnie program korzysta z wywołania MQINQ w celu sprawdzenia wartości atrybutów *InhibitGet*, **CurrentQDepth**i **OpenInputCount** w kolejce docelowej.

Jeśli wywołanie MQINQ powiedzie się, program korzysta z wywołania MQPUT1 w celu umieszczenia komunikatu odpowiedzi w kolejce odpowiedzi. Ten komunikat zawiera wartości trzech atrybutów.

Jeśli wywołanie MQOPEN lub MQINQ nie powiodło się, program korzysta z wywołania MQPUT1 w celu umieszczenia komunikatu raportu w kolejce odpowiedzi. W polu *Feedback* deskryptora komunikatu tego komunikatu raportu jest to kod przyczyny zwracany przez wywołanie MQOPEN lub MQINQ, w zależności od tego, który błąd nie powiódł się.

Po wywołaniu wywołania MQINQ program zamknie kolejkę docelową przy użyciu wywołania MQCLOSE.

Jeśli w kolejce żądań nie pozostały żadne komunikaty, program zamknie tę kolejkę i rozłącza się z menedżerem kolejek.

Właściwości Inquire przykładowego programu Message Handle

AMQSIQMA to przykładowy program w języku C do sprawdzania właściwości uchwytu komunikatu z kolejki komunikatów. Jest to przykład użycia wywołania funkcji API MQINQMP.

W tym przykładzie tworzony jest uchwyt komunikatu i umieszcza go w polu MsgHandle w strukturze MQGMO. Następnie próbka pobiera jeden komunikat i pobiera i drukuje wszystkie właściwości, z którymi uchwyt komunikatu został zapełniony.

```
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
```

Programy przykładowe publikowania/subskrypcji

Przykładowe programy publikowania/subskrypcji demonstrują sposób użycia funkcji publikowania i subskrypcji w produkcie IBM MQ.

Istnieją trzy przykładowe programy w języku C ilustrujące sposób działania programu w interfejsie publikowania/subskrybowania produktu IBM MQ . Istnieje kilka przykładów języka C, które korzystają ze starszych interfejsów, a także przykłady produktu Java . Przykłady produktu Java korzystają z interfejsu publikowania/subskrypcji produktu IBM MQ w pliku com.ibm.mq.jar i interfejsu publikowania/subskrypcji produktu JMS w pliku com.ibm.mqjms. Przykłady JMS nie są omówione w tym temacie.

C

Znajdź przykładowy publikator `amqspub` w folderze przykładów produktu `C` . Uruchom go z dowolną nazwą tematu, którą lubisz jako pierwszy, a następnie opcjonalną nazwą menedżera kolejek. Na przykład: `amqspub mytopic QM3` . Istnieje również wersja klienta o nazwie `amqspubc`. Jeśli zostanie wybrana opcja uruchamiania wersji klienta, należy najpierw zapoznać się z informacjami szczegółowymi “Konfigurowanie menedżera kolejek w celu akceptowania połączeń klientów na wielu platformach” na stronie 1177 .

Publikator łączy się z domyślnym menedżerem kolejek i odpowiada na dane wyjściowe, `target topic is mytopic` . Każdy wiersz wprowadzany do tego okna od teraz jest publikowany w produkcie `mytopic` .

Otwórz inne okno komend w tym samym katalogu, a następnie uruchom program subskrybenta `amqssub`, podając tę samą nazwę tematu i opcjonalną nazwę menedżera kolejek. Na przykład: `amqssub mytopic QM3` .

Subskrybent odpowiada za pomocą danych wyjściowych `Calling MQGET : 30 seconds wait time` . Od tej pory w danych wyjściowych subskrybenta pojawiają się wiersze, które wpisujesz do publikatora.

Uruchom innego subskrybenta w innym oknie komend i obserwuj oba subskrybenty, które otrzymują publikacje.

Pełną dokumentację parametrów, w tym opcje ustawień, można znaleźć w przykładowym kodzie źródłowym. Wartości w polu opcji subskrybenta zostały opisane w następującym temacie: Opcje (MQLONG).

Istnieje inny przykładowy subskrybent `amqssbx`, który oferuje dodatkowe opcje subskrypcji jako przełączniki wiersza komend.

Wpisz `amqssbx -d mysub -t mytopic -k` , aby wywołać subskrybenta 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 kilka elementów w tym samym temacie. Zrestartuj subskrybenta. Ostatni element opublikowany w czasie, gdy subskrybent nie był uruchomiony, jest wyświetlany przez subskrybent natychmiast po jego zrestartowaniu.

Wcześniejsza wersja

Istnieje dodatkowy zestaw przykładów języka C, które demonstrują komendy w kolejce. Niektóre z tych przykładów zostały oryginalnie wysłane jako część pakietu MQ0C Supportpac. Możliwości demonstrować są w pełni obsługiwane, ze względu na kompatybilność.

Zniechęcamy Cię do korzystania z interfejsu komend w kolejce. Jest on znacznie bardziej złożony niż interfejs API publikowania/subskrypcji i nie ma przekonujących powodów funkcjonalnych do programowania złożonych komend w kolejce. Jednak podejście w kolejce może być bardziej odpowiednie, być może dlatego, że korzystasz już z interfejsu, lub ponieważ środowisko programistyczne ułatwia budowanie złożonego komunikatu i wywoła ogólną operację MQPUT, a nie konstruowanie różnych wywołań MQSUB.

Dodatkowe przykłady znajdują się w podkatalogu `pubsub` w folderze `samples` .

W produkcie Tabela 170 na stronie 1208znajduje się sześć typów przykładów.

Tabela 170. Kategorie wcześniejszych przykładowych programów w języku C publikowania/subskrypcji

Kategoria	Programy	Komentarze
RFH1	`amqssr1a.c` `amqspr1a.c`	Prosty przykład publikowania/subskrypcji zbudowany przy użyciu komunikatów w formacie RFH1 .
RFH2	`amqssr2a.c` `amqspr2a.c`	Prosty przykład publikowania/subskrypcji zbudowany przy użyciu komunikatów w formacie RFH2 .
Przykłady MQAI	`amqsppca.c` `amqsspca.c`	Prosty przykład publikowania/subskrypcji zbudowany za pomocą komend PCF i interfejsu komend MQAI.
MA0C Wyniki usługi przy użyciu RFH1	`amqsgama.c` `amqsresa.c`	Usługa wyników zbudowana przy użyciu nagłówków RFH1 1. Wymaga kolejek zdefiniowanych w `amqsgama.tst` i `amqsresa.tst` 2. Produkt `amqsresa` musi być uruchomiony przed `amqsgama`
MA0C Wyniki usługi przy użyciu RFH2	`amqsgr2a.c` `amqsrr2a.c`	Usługa wyników zbudowana przy użyciu nagłówków RFH2 1. Wymaga kolejek zdefiniowanych w `amqsgama.tst` i `amqsresa.tst` 2. Produkt `amqsresa` musi być uruchomiony przed `amqsgama`
Przykład publikowania/ subskrybowania wyjścia routingu	`amqspsra.c`	Demonstruje, jak zmienić miejsce docelowe kolejki lub menedżera kolejek dla komunikatu publikowania/subskrypcji w wyjściu routingu.

Przykładowy program dla produktu Java

The Java sample `MQPubSubApiSample.java` combines publisher and subscribers in a single program. Jego źródłowe i skompilowane pliki klas znajdują się w folderze przykładów produktu `wmqjava` .

Jeśli zostanie wybrana opcja uruchamiania w trybie klienta, należy najpierw zapoznać się z informacjami szczegółowymi “Konfigurowanie menedżera kolejek w celu akceptowania połączeń klientów na wielu platformach” na stronie 1177 .

Uruchom przykład z wiersza komend za pomocą komendy Java , jeśli skonfigurowano środowisko Java . Przykład można uruchomić z poziomu obszaru roboczego programu IBM MQ Explorer Eclipse , który zawiera już skonfigurowane środowisko robocze programistyczne produktu Java .

W celu jego uruchomienia może być konieczna zmiana niektórych właściwości programu przykładowego. Można to zrobić, podając parametry dla maszyny JVM lub edytując źródło.

Instrukcje zawarte w sekcji "Uruchamianie przykładu MQPubSubApiSample Java" na stronie 1209 przedstawiają sposób uruchamiania przykładu z obszaru roboczego Eclipse .

Uruchamianie przykładu MQPubSubApiSample Java
Sposób uruchamiania produktu MQPubSubApiSample przy użyciu narzędzi programistycznych produktu Java z poziomu platformy Eclipse .

Zanim rozpoczniesz

Otwórz środowisko robocze Eclipse . Utwórz nowy katalog obszaru roboczego i wybierz go. Zamknij okno powitalne.

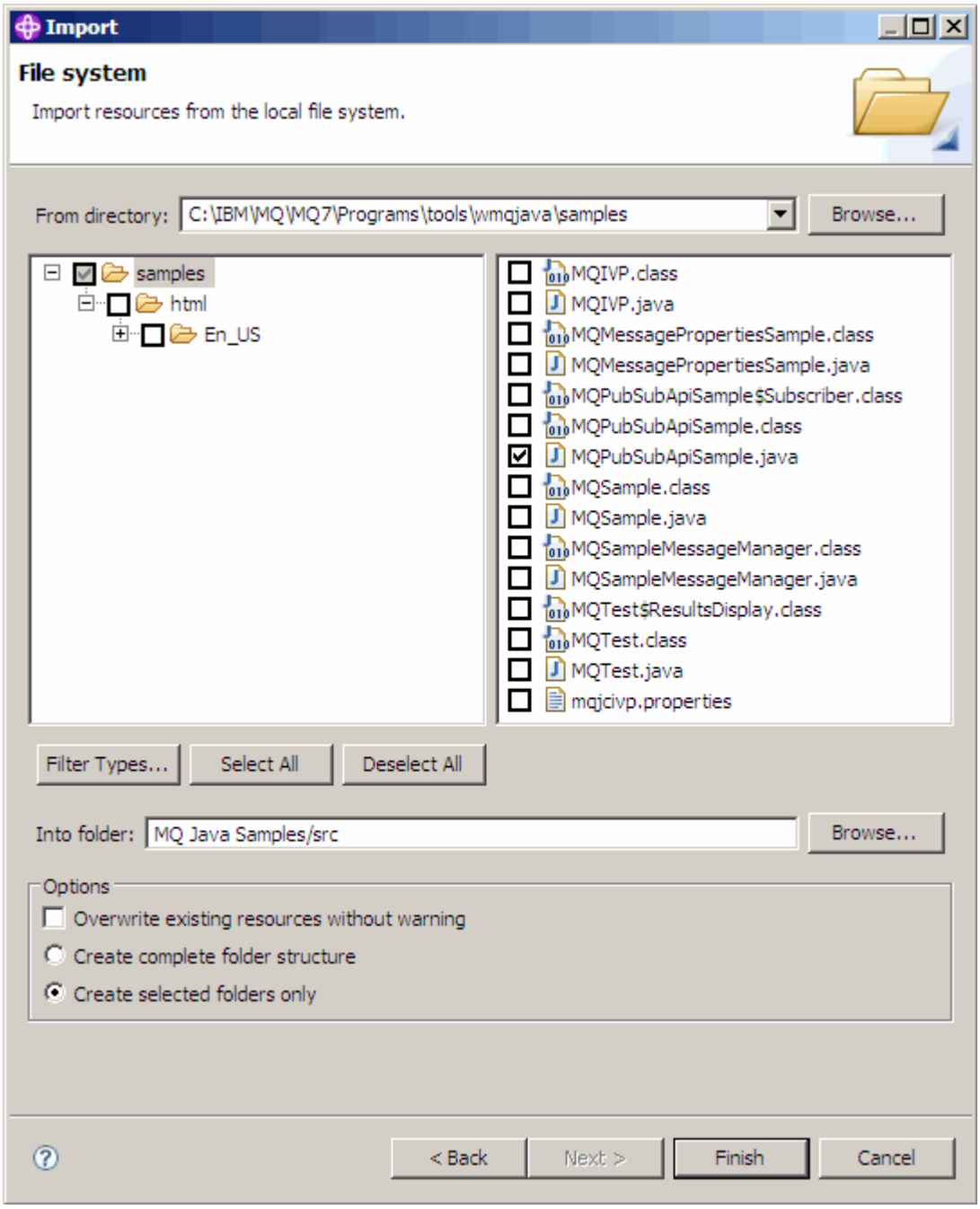
Wykonaj kroki opisane w sekcji "Konfigurowanie menedżera kolejek w celu akceptowania połączeń klientów na wielu platformach" na stronie 1177 przed uruchomieniem jako klient.

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 nasłuchiwania na porcie 1414. Zadanie opisuje ten prosty przypadek i zawiera ogólne informacje dotyczące sposobu udostępniania parametrów i modyfikowania przykładu w celu dopasowania do różnych konfiguracji produktu IBM MQ . Przykład jest ilustrowany uruchomionym na serwerze Windows. Ścieżki do plików będą się różnić na innych platformach.

Procedura

1. Importowanie przykładowych programów produktu 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 białą przestrzeń w widoku **Eksplorator pakietów** . Kliknij opcję **Nowy** > **ProjektJava**.
   d) W polu **`Project name`** wpisz `MQ Java Samples`. Kliknij przycisk **Dalej**.
   e) Na panelu **`Java Settings`** przejdź do karty **Libraries** (Biblioteki).
   f) Kliknij opcję **Dodaj zewnętrzne pliki JAR**.
   g) Przejdź do katalogu *MQ_INSTALLATION_PATH* `\java\lib` , gdzie *MQ_INSTALLATION_PATH* jest folderem instalacyjnym produktu IBM MQ , a następnie wybierz opcję `com.ibm.mq.jar` i `com.ibm.mq.jmqi.jar` .
   h) Kliknij opcję **Otwórz** > **Zakończ.**
   i) Kliknij prawym przyciskiem myszy `src` w widoku **Eksplorator pakietów** .
   j) Wybierz opcję **Importuj ...** > **Ogólne** > **System plików** > **Dalej** > **Przeglądaj**... i przejdź do ścieżki *MQ_INSTALLATION_PATH* `\tools\wmqjava\samples` , gdzie *MQ_INSTALLATION_PATH* jest katalogiem instalacyjnym IBM MQ .
   k) Na panelu **Importuj** (Rysunek 135 na stronie 1210) kliknij przycisk `samples` (nie zaznaczaj pola wyboru).
   l) Wybierz opcję `MQPubSubApiSample.java`. Pole **`Into folder`** powinno zawierać `MQ Java Samples/src`. Kliknij przycisk **Zakończ**.



Rysunek 135. 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 uruchamia program bez dokonywania żadnych zmian:
     - W menu głównym obszaru roboczego rozwiń folder `src` (Menu główne). Kliknij **MQPubSubApiSample.java** przyciskiem **Uruchom jako** > **1. Aplikacja Java**
   - Drugi wybór uruchamia program z parametrami lub z zmodyfikowanym kodem źródłowym dla Twojego środowiska:
     - Otwórz program `MQPubSubApiSample.java` i przestudiuj konstruktor `MQPubSubApiSample` .

– Zmodyfikuj atrybuty programu.

  These attributes are modifiable using the -D JVM switch, or by providing a default value for the System property by editing the source code.

  - topicObject
  - QueueManagerName
  - subscriberCount

  Te atrybuty mogą być zmieniane tylko przez edycję kodu źródłowego w konstruktorze.

  - nazwa hosta
  - port
  - kanał

  Aby ustawić System properties, należy zakodować wartość domyślną w akcesorze, na przykład:

```
queueManagerName = System.getProperty("com.ibm.mq.pubSubSample.queueManagerName",
"QM3");
```

  Lub podaj parametr dla maszyny JVM za pomocą opcji `-D` , tak jak pokazano to w poniższych 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 w obszarze **Utwórz, zarządzaj i uruchom aplikacje** , a następnie kliknij kartę **(x) = Argumenty** .
c. W panelu **Argumenty maszyny VM:** wpisz `-D` i wklej nazwę System.property , `com.ibm.mq.pubSubSample.queueManagerName`, a następnie `=QM3`. Kliknij kolejno opcje **Zastosuj** > **Uruchom**.
d. Dodaj kolejne argumenty jako listę rozdzielaną przecinkami lub jako dodatkowe wiersze w panelu bez separatorów 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 wyjścia w celu przechwycenia publikacji, zanim zostanie dostarczony do subskrybenta. Następnie wyjście może na przykład zmienić nagłówki komunikatu, ładunek lub miejsce docelowe albo uniemożliwić opublikowanie komunikatu w subskrybencie.

Aby uruchomić przykład, wykonaj następujące czynności:

1. Skonfiguruj menedżer kolejek:
   - Linux UNIX W systemach UNIX and Linux dodaj sekcję taką, jak ta, do pliku `qm.ini` :

```
PublishSubscribe:
PublishExitPath=Module
PublishExitFunction=EntryPoint
```

     gdzie moduł jest *MQ_INSTALLATION_PATH*`/samp/bin/amqspse`. *MQ_INSTALLATION_PATH* reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .
   - Windows W systemie Windows ustaw równoważne atrybuty w rejestrze.
2. Upewnij się, że moduł jest dostępny dla produktu IBM MQ.
3. Zrestartuj menedżer kolejek, aby odebrać konfigurację.

4. W procesie aplikacji, który ma być śledzony, należy opisać miejsce, w którym powinny być zapisywane pliki śledzenia. Na przykład:
   - Linux UNIX W systemach UNIX and Linux upewnij się, że katalog `/var/mqm/trace` istnieje i wyeksportuj następującą zmienną środowiskową:

     ```
     export MQPSE_TRACE_LOGFILE=/var/mqm/trace/PubTrace
     ```

   - Windows W systemie Windowsupewnij się, że katalog `C:\temp` istnieje i ustaw następującą zmienną środowiskową:

     ```
     set MQPSE_TRACE_LOGFILE=C:\temp\PubTrace
     ```

Przykładowe programy umieszczania

Programy przykładowe umieszczone w kolejce umieszczają komunikaty w kolejce przy użyciu wywołania MQPUT.

Nazwy tych programów można znaleźć w sekcji “Funkcje demonstruje w przykładowych programach na Multiplatforms” na stronie 1168 .

Projekt przykładowego programu "Put"

Program korzysta z wywołania MQOPEN z opcją MQOO_OUTPUT, aby otworzyć kolejkę docelową w celu umieszczania komunikatów.

Jeśli nie jest możliwe otwarcie kolejki, program wyświetli komunikat o błędzie zawierający kod przyczyny zwrócony przez wywołanie MQOPEN. Aby program był prosty, w tym i w kolejnych wywołaniach MQI program używa wartości domyślnych dla wielu opcji.

Dla każdego wiersza danych wejściowych program odczytuje tekst do buforu i korzysta z wywołania MQPUT w celu utworzenia komunikatu datagramu zawierającego tekst tego wiersza. Program będzie kontynuowany aż do momentu, gdy dojdzie do końca danych wejściowych lub wywołanie MQPUT nie powiedzie się. Jeśli program dociera do końca danych wejściowych, zamyka kolejkę za pomocą wywołania MQCLOSE.

Uruchamianie przykładowych programów Put

Uruchamianie przykładów amqsput i amqsputc

ULW

Przykład amqqsput jest programem do umieszczania komunikatów przy użyciu powiązań lokalnych, a przykład amqsputc jest programem do umieszczania przy użyciu powiązań klienta. Każdy z tych programów ma następujące parametry pozycyjne:

1. Nazwa kolejki docelowej (wymagane)
2. Nazwa menedżera kolejek (opcjonalna)

   Jeśli nie określono menedżera kolejek, komenda amqsput nawiązuje połączenie z domyślnym menedżerem kolejek, a komenda 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.

Programy te używają również zmiennej środowiskowej o nazwie **MQSAMP_USER_ID** , która powinna być ustawiona na identyfikator użytkownika używany do uwierzytelniania połączenia. Jeśli ta opcja jest ustawiona, program poprosi o podanie hasła, które ma towarzyszyć temu identyfikatorowi użytkownika.

Aby uruchomić te programy, wprowadź jedną z następujących komend:

- `amqsput myqueue qmanagername`
- `amqsputc myqueue qmanagername`

gdzie `myqueue` jest nazwą kolejki, w której zostaną umieszczone komunikaty, a `qmanagername` jest menedżerem kolejek, który jest właścicielem produktu `myqueue`.

Uruchamianie przykładu amq0put

ULW

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 Message Message

Przykłady komunikatów referencyjnych umożliwiają przesyłanie dużego obiektu z jednego węzła do innego (zwykle w różnych systemach) bez konieczności przechowywania obiektu w kolejkach produktu IBM MQ w węźle źródłowym lub docelowym.

Udostępniono zestaw przykładowych programów w celu zademonstrować, w jaki sposób komunikaty odniesienia mogą być umieszczane w kolejce, odbierane przez wyjścia komunikatów i pobierane z kolejki. Programy przykładowe korzystają z komunikatów referencyjnych do przenoszenia plików. Jeśli chcesz przenieść inne obiekty, takie jak bazy danych, lub jeśli chcesz wykonać sprawdzenia bezpieczeństwa, zdefiniuj własne wyjście, na podstawie przykładu, amqsxrm.

Wersja przykładowego programu obsługi wyjścia komunikatów odniesienia zależy od platformy, na której jest uruchomiony kanał:

- Na wszystkich platformach użyj amqsxrma na końcu wysyłania.
- Użyj amqsxrma na końcu odbioru, jeśli odbiornik jest uruchomiony na dowolnej platformie z wyjątkiem IBM i.
- IBM i Jeśli odbiornik działa pod kontrolą systemu IBM i, należy użyć komendy amqsxrm4.

.

IBM i *Uwagi dotyczące użytkowników produktu IBM i*

Aby otrzymać komunikat referencyjny za pomocą przykładowego wyjścia komunikatu, należy określić plik w głównym systemie plików IFS lub dowolnym podkatalogu, aby można było utworzyć plik strumieniowy.

Przykładowe wyjście komunikatów na serwerze IBM i tworzy plik, przekształca dane w kod EBCDIC i ustawia stronę kodową na stronę kodową systemu. Następnie można skopiować ten plik do katalogu QSYS.LIB w systemie plików 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 pliku. Przed uruchomieniem tej komendy należy ją utworzyć.

Jeśli plik został wysłany z biblioteki QSYS.LIB, nie są wymagane żadne zmiany w próbkach. W przypadku każdego innego systemu plików identyfikator CCSID określony w polu CodedCharSetId w strukturze MQRMH jest zgodny z danymi masowymi, które są wysyłane.

W przypadku korzystania z zintegrowanego systemu plików należy utworzyć moduły programu z ustawioną opcją SYSIFCOPT (*IFSIO). Aby przenieść bazę danych lub pliki rekordów o stałej długości, należy zdefiniować własne wyjście w oparciu o dostarczonej przykładowej AMQSXRM4.

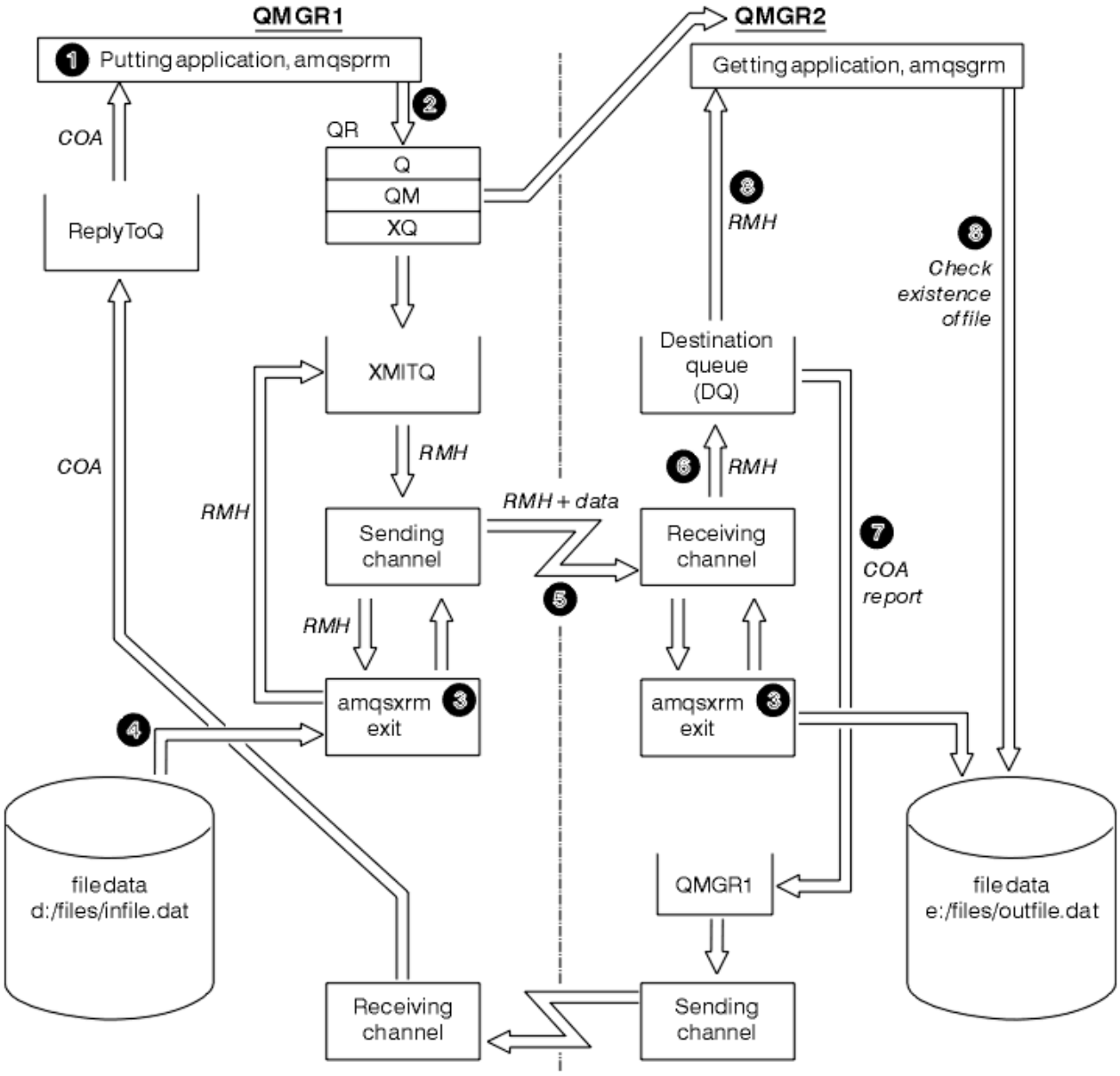
Zalecaną metodą przesyłania zbioru bazy danych jest przekształcenie go w strukturę IFS za pomocą komendy CPYTOSTMF, a następnie wysłanie komunikatu odwołania dołączającego do zbioru IFS. Jeśli zostanie wybrana opcja przesyłania zbioru bazy danych przez odwołanie się do niego z poziomu systemu

plików IFS, ale nie przekształć go w strukturę IFS, należy określić nazwę podzbioru. Integralność danych nie jest zagwarantowana, jeśli zostanie wybrana ta metoda.

Multi *Uruchamianie przykładów komunikatów odniesienia*

W tym przykładzie przedstawiono sposób uruchamiania przykładowej aplikacji AMQSPRM komunikatów odniesienia w systemach UNIX, Linuxi Windows, lub AMQSPRMA w systemie IBM i. W przykładzie pokazano, w jaki sposób można umieścić komunikaty referencyjne w kolejce, odbierane przez wyjścia komunikatów i pobierane z kolejki.

Przykłady komunikatów odniesienia są uruchamiane w następujący sposób:



Rysunek 136. Uruchamianie przykładów komunikatów odniesienia

1. Skonfiguruj środowisko, aby uruchomić programy nasłuchujące, kanały i monitory wyzwalacza, a także zdefiniuj kanały i kolejki.

   W celu opisania sposobu konfigurowania komunikatu referencyjnego, ten przykład odnosi się do maszyny wysyłającej jako MACHINE1 z menedżerem kolejek o nazwie QMGR1 i maszyną odbierającą jako MACHINE2 z menedżerem kolejek o nazwie QMGR2.

Uwaga: Następujące definicje umożliwiają budowanie komunikatu referencyjnego w celu wysłania pliku z typem obiektu FLATFILE z menedżera kolejek QMGR1 do QMGR2 i ponowne utworzenie pliku zgodnie z definicją w wywołaniu funkcji AMQSPRM (lub AMQSPRMA w systemie IBM i). Komunikat referencyjny (wraz z danymi pliku) jest wysyłany przy użyciu kanału CHL1 i kolejki transmisji XMITQ i umieszczany w kolejce DQ. Raporty o wyjątkach i COA są wysyłane z powrotem do QMGR1 za pomocą kanału REPORT i kolejki transmisji QMGR1.

Aplikacja, która odbiera komunikat Reference (AMQSGRM lub AMQSGRMA w systemie IBM i) jest wyzwalana przy użyciu kolejki inicjuj 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 zainstalowano produkt IBM MQ .

IBM i Definicje MQSC używane są w stylu AIX do definiowania wyjść, dlatego w przypadku korzystania z komend MQSC w produkcie IBM inaleźy odpowiednio zmodyfikować te definicje. Należy pamiętać, że w danych komunikatu FLATFILE jest rozróżniana wielkość liter, a próba nie będzie działać, jeśli nie jest zapisana wielkimi literami.

Na komputerze MACHINE1, menedżer kolejek QMGR1

Składnia 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 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 Składnia komendy IBM i

Uwaga: IBM i Jeśli w systemie IBM inie zostanie określona 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 nasłuchiwanie dla menedżerów kolejek wysyłających i odbierających
   b. Uruchom kanały CHL1 i REPORT
   c. W odbiorczym menedżerze kolejek uruchom monitor wyzwalacza dla kolejki inicjującej INITQ.
3. W wierszu komend wywołaj przykładowy program put Reference Message AMQSPRM (AMQSPRMA on IBM i) , korzystając z następujących parametrów:

   -m
   Nazwa lokalnego menedżera kolejek. Domyślnie jest 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 znajduje się kolejka zdefiniowana w parametrze -q. Wartością domyślną jest menedżer kolejek określony w parametrze -m.

   -t
   Typ obiektu

   -w
   Interwał oczekiwania, czyli czas oczekiwania na zgłoszenia wyjątków i COA z menedżera kolejek odbierających

   Na przykład, aby użyć przykładu z obiektami zdefiniowanymi wcześniej, należy użyć następujących parametrów:

```
-mQMGR1 -iInput File -oOutput File -qQR -tFLATFILE -w120
```

   Wydłużanie czasu oczekiwania pozwala na wysłanie dużego pliku w sieci przed wyświetleniem limitu czasu przez program.

```
amqsprm -q QR -m QMGR1 -i d:\x\file.in -o d:\y\file.out -t FLATFILE
```

Użytkownicy systemu IBM i: 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 to, 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 za pomocą komendy CRTDIR, a nie używając katalogu głównego.

c. W przypadku wywołania 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 iprzy określaniu 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 UNIX W przypadku platform UNIX and Linux należy użyć dwóch ukośników odwrotnych (\\) zamiast jednego, aby oznaczyć docelowy katalog plików. Oznacza to, że 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 Message Message powoduje, że:

- Komunikat odniesienia jest umieszczany w kolejce QR w menedżerze kolejek QMGR1.
- Plik źródłowy i ścieżka są `d:\files\infile.dat` i istnieją w systemie, w którym została wydana przykładowa komenda.
- Jeśli kolejka QR jest kolejką zdalną, komunikat odniesienia jest wysyłany do innego menedżera kolejek w innym systemie, w którym tworzony jest plik o nazwie i ścieżce `e:\files\outfile.dat`. Zawartość tego pliku jest taka sama, jak w pliku źródłowym.
- amqsprm czeka przez 30 sekund na raport COA z docelowego menedżera kolejek.
- Typ obiektu to `flatfile`, więc kanał używany do przenoszenia komunikatów z kolejki QR musi określać to pole w polu *MsgData* .

4. Podczas definiowania kanałów należy wybrać wyjście komunikatów zarówno w wysyłającym, jak i odbierającym końcach, aby był on amqsxrm.

Windows Wartość ta jest zdefiniowana w systemie Windows w następujący sposób:

```
msgexit(' pathname\amqsxrm.dll(MsgExit)')
```

Solaris AIX Wartość ta jest zdefiniowana w następujących systemach: AIX i Solaris :

```
msgexit(' pathname/amqsxrm(MsgExit)')
```

Jeśli określiłeś nazwę ścieżki, podaj pełną nazwę. Jeśli nazwa ścieżki zostanie pominięta, to przyjmuje się, że program znajduje się w ścieżce określonej w pliku `qm.ini` (lub w katalogu IBM MQ for Windows, w ścieżce określonej w rejestrze).

5. Wyjście kanału odczytuje nagłówek komunikatu odwołania i znajduje plik, do którego odwołuje się ten nagłówek.
6. Wyjście kanału może następnie segmentować plik przed wysłaniem go w dół kanału wraz z nagłówkiem.

   Solaris AIX W systemach AIX i Solariszmień właściciela grupy docelowej na 'mqm', aby program obsługi wyjścia komunikatów mógł utworzyć plik w tym katalogu. Zmień również uprawnienia katalogu docelowego, aby zezwolić członkom grupy mqm na zapisywanie w tym katalogu. Dane pliku nie są zapisywane w kolejkach produktu IBM MQ .
7. Gdy ostatni segment zbioru jest przetwarzany przez wyjście komunikatu odbierającego, komunikat odniesienia jest umieszczany w kolejce docelowej określonej przez amqsprm. Jeśli ta kolejka jest wyzwalana (oznacza to, że definicja określa atrybuty kolejki **Trigger**, **InitQ**i **Process**), wyzwalany jest program określony przez parametr PROC kolejki docelowej. Program, który ma zostać wyzwolony, musi być zdefiniowany w polu `ApplId` atrybutu **Process** .
8. Gdy komunikat referencyjny osiągnie kolejkę docelową (DQ), raport COA jest przesyłany z powrotem do aplikacji umieszczanie (amqsprm).
9. Przykładowy komunikat Get Reference Message, amqsgrm, pobiera komunikaty z kolejki określonej w komunikacie wyzwalacza wejściowego i sprawdza, czy plik istnieje.

Projekt przykładowego komunikatu umieszczonego w odwołaniu (amqsprma.c, AMQSPRM4)
W tym temacie przedstawiono szczegółowy opis przykładowego komunikatu umieszczonego w odwołaniu.

W tym przykładzie tworzony jest komunikat referencyjny, który odwołuje się do pliku i umieszcza go w określonej kolejce:

1. Przykład łączy się z lokalnym menedżerem kolejek za pomocą MQCONN.
2. Następnie zostanie otwarta (MQOPEN) kolejka modelowa, która jest używana do odbierania komunikatów raportu.
3. W tym przykładzie tworzony jest komunikat referencyjny zawierający wartości wymagane do przeniesienia pliku, na przykład nazwy plików źródłowych i docelowych oraz typ obiektu. Przykład: próbka dostarczana z produktem IBM MQ buduje komunikat referencyjny, aby wysłać plik `d:\x\file.in` z `QMGR1` do `QMGR2` i ponownie utworzyć plik jako `d:\y\file.out` , korzystając z 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 UNIX and Linux należy użyć dwóch ukośników odwrotnych (\\) zamiast jednego, aby oznaczyć docelowy katalog plików. Oznacza to, że komenda **amqsprm** wygląda następująco:

   ```
   amqsprm -q QR -m QMGR1 -i /x/file.in -o d:\\y\\file.out -t FLATFILE
   ```
4. Komunikat referencyjny jest umieszczany (bez żadnych danych) do kolejki określonej przez parametr /q. Jeśli jest to kolejka zdalna, komunikat jest umieszczany w odpowiedniej kolejce transmisji.
5. Próbka czeka, przez czas określony w parametrze wagowym (domyślnie 15 sekund), dla raportów COA, które wraz z raportami wyjątków są wysyłane z powrotem do kolejki dynamicznej utworzonej w lokalnym menedżerze kolejek (QMGR1).

Projekt przykładu wyjścia komunikatu odniesienia (amqsxrma.c, AMQSXRM4)
W tym przykładzie rozpoznawane są komunikaty odniesienia z typem obiektu zgodnym z typem obiektu w polu danych użytkownika wyjścia komunikatu definicji kanału.

W przypadku tych komunikatów wykonywane są następujące działania:

- W kanale nadawczym lub kanale serwera określona długość danych jest kopiowana z określonego przesunięcia określonego pliku do obszaru pozostałego w buforze agenta po komunikacie odniesienia. Jeśli koniec pliku nie zostanie osiągnięty, po zaktualizowaniu pola *DataLogicalOffset* zostanie ponownie umieszczony komunikat odniesienia w kolejce transmisji.
- W kanale requestera lub odbiornika, jeśli pole *DataLogicalOffset* ma wartość zero, a podany plik nie istnieje, zostanie utworzony. Dane następujące po komunikacie referencyjnym są dodawane na końcu określonego pliku. Jeśli komunikat odniesienia nie jest ostatnim komunikatem dla określonego pliku, zostanie on usunięty. W przeciwnym razie zostanie on zwrócony do wyjścia kanału bez dołączonych danych, które mają zostać umieszczone w kolejce docelowej.

W przypadku kanałów nadawcy i serwera, jeśli pole *DataLogicalLength* w wejściowym komunikacie odniesienia jest zerowe, pozostała część pliku, od *DataLogicalOffset* do końca pliku, ma być wysłana wzdłuż kanału. Jeśli wartość nie jest równa zero, wysyłana jest tylko określona długość.

Jeśli wystąpi błąd (na przykład, jeśli próbka nie może otworzyć pliku), MQCXP. Parametr *ExitResponse* jest ustawiony na wartość MQXCC_SUPPRESS_FUNCTION w taki sposób, że przetwarzany komunikat jest umieszczany w kolejce niedostarczonych komunikatów, a nie w kolejce docelowej. W MQCXP zwracany jest kod sprzężenia zwrotnego. *Feedback* i zwróć do aplikacji, która wstawiła komunikat w polu *Feedback* deskryptora komunikatu komunikatu raportu. Wynika to z faktu, że aplikacja żądała raportów o wyjątkach, ustawiając parametr MQRO_EXCEPTION w polu *Report* deskryptora MQMD.

Jeśli kodowanie lub *CodedCharacterSetId* (CCSID) komunikatu odwołania różni się od wartości w menedżerze kolejek, komunikat odniesienia jest przekształcany w kodowanie lokalne i identyfikator CCSID. W naszym przykładzie, amqsprm, formatem obiektu jest MQFMT_STRING, więc amqsxrm przekształca dane obiektu na lokalny identyfikator CCSID w odbierającym końcu, zanim dane zostaną zapisane do pliku.

Nie należy określać formatu przesyłanego pliku jako MQFMT_STRING, jeśli plik zawiera znaki wielobajtowe (na przykład DBCS lub Unicode). Wynika to z faktu, że znak wielobajtowy może zostać podzielony, gdy plik jest podzielony na segmenty podczas wysyłania. Aby przesłać i przekształcić taki plik, należy określić format jako inny niż MQFMT_STRING w taki sposób, aby program obsługi wyjścia komunikatu odwołania nie konwertuje go i konwertuje plik na odbiorczy koniec po zakończeniu przesyłania.

Kompilowanie przykładu Wyjście z komunikatu odniesienia
Aby skompilować przykład wyjścia komunikatów odniesienia, należy użyć komendy dla platformy, na której zainstalowany jest produkt IBM MQ .

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest 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 używał on systemu plików IFS, dodaj opcję `SYSIFCOPT(*IFSIO)` .
2. Aby utworzyć program do użycia z kanałami innymi niż wielowątkowe, należy użyć następującej komendy: `CRTPGM PGM(MYLIB/AMQSXRMA) BNDSRVPGM(QMQM/LIBMQM)`
3. Aby utworzyć program do użycia z kanałami wielowątkowym, 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.Solaris

Solaris

```
$ cc -xarch=v9 -mt -G -o /var/mqm/exits64/amqsxrma amqsqrma.c -I  MQ_INSTALLATION_PATH/inc
-L  MQ_INSTALLATION_PATH/lib64 -R  MQ_INSTALLATION_PATH/lib64 -R/usr/lib/64 -lmqm


-lsocket
-lnsl -ldl
```

wł.Windows

Windows Produkt IBM MQ udostępnia teraz bibliotekę mqm z pakietami klientów, a także pakiety serwera, dlatego w poniższym przykładzie zamiast produktu `mqmvx.lib`jest używany produkt `mqm.lib` :

```
cl amqsqrma.c /link /out:amqsxrm.dll /dll mqm.lib mqm.lib /def:amqsxrm.def
```

Pojęcia pokrewne

“Pisanie programów obsługi wyjścia kanału” na stronie 1062
W celu ułatwienia pisania programów obsługi wyjścia kanału można użyć następujących informacji.

Projekt przykładowego komunikatu Get Reference Message (amqsgrma.c, AMQSGRM4)
W tym temacie opisano sposób projektowania przykładowego komunikatu Get Reference Message (Pobierz komunikat odniesienia).

Logika programu jest następująca:

1. Próbka jest wyzwalana i wyodrębnia nazwy kolejek i menedżerów kolejek z wejściowego komunikatu wyzwalacza.
2. Następnie łączy się on z określonym menedżerem kolejek za pomocą MQCONN i otwiera określoną kolejkę za pomocą komendy MQOPEN.
3. Przykładowe problemy MQGET z odstępem czasu oczekiwania na 15 sekund w pętli, aby pobrać komunikaty z kolejki.

4. Jeśli komunikat jest komunikatem odniesienia, próbka sprawdza istnienie pliku, który został przesłany.
5. Następnie zamyka kolejkę i rozłącza się z menedżerem kolejek.

Przykładowe programy żądania

Przykładowe programy Request demonstrują przetwarzanie klient/serwer. Przykłady są to klienty, które umieszczają komunikaty żądań w docelowej kolejce serwera, która jest przetwarzana przez program serwera. Oczekują na to, że program serwera umieje umieścić komunikat odpowiedzi w kolejce odpowiedzi.

Przykłady żądań umieszczają serię komunikatów żądań w docelowej kolejce serwera przy użyciu wywołania MQPUT. Komunikaty te określają kolejkę lokalną SYSTEM.SAMPLE.REPLY jako kolejkę odpowiedzi, która może być kolejką lokalną lub zdalną. Programy oczekują na komunikaty odpowiedzi, a następnie wyświetlają je. Odpowiedzi są wysyłane tylko wtedy, gdy docelowa kolejka serwera jest przetwarzana przez aplikację serwera lub jeśli aplikacja jest wyzwalana w tym celu (programy przykładowe Inkwire, Set i Echo zostały zaprojektowane do wyzwolenia). Próbka C czeka 1 minuta (próbka języka COBOL czeka 5 minut), pierwsza odpowiedź na przyjazd (w celu umożliwienia wyzwolenia aplikacji serwera) oraz 15 sekund na kolejne odpowiedzi, ale obie próbki mogą zakończyć się bez uzyskania odpowiedzi. Nazwy programów przykładowych żądań można znaleźć w sekcji “Funkcje demonstruje w przykładowych programach na Multiplatforms” na stronie 1168 .

Uruchamianie przykładowych programów żądania

Uruchamianie przykładów amqsreq0.c, amqsreq i amqsreqc

W wersji C programu przyjmuje się trzy parametry:

1. Nazwa docelowej kolejki serwera (konieczne)
2. Nazwa menedżera kolejek (opcjonalnie)
3. Kolejka odpowiedzi (opcjonalnie)

Na przykład wprowadź jedną z następujących opcji:

- `amqsreq myqueue qmanagername replyqueue`
- `amqsreqc myqueue qmanagername`
- `amq0req0 myqueue`

gdzie `myqueue` to nazwa docelowej kolejki serwera, `qmanagername` to nazwa menedżera kolejek, który jest właścicielem `myqueue`, a `replyqueue` to nazwa kolejki odpowiedzi.

Jeśli nazwa menedżera kolejek zostanie pominięta, to przyjmuje się, że domyślny menedżer kolejek jest właścicielem kolejki. Jeśli nazwa kolejki odpowiedzi zostanie pominięta, zostanie podana domyślna kolejka odpowiedzi.

Uruchamianie przykładu amq0req0.cbl

Wersja COBOL nie ma żadnych parametrów. Jest on połączony z domyślnym menedżerem kolejek i po uruchomieniu zostanie wyświetlony monit:

```
Please enter the name of the target server queue
```

Program pobiera dane wejściowe z komendy StdIn i dodaje każdy wiersz do docelowej kolejki serwera, przyjmując każdy wiersz tekstu jako treść komunikatu żądania. Program kończy się, gdy odczytywa się wiersz o wartości NULL.

Uruchamianie przykładu AMQSREQ4

Program C tworzy komunikaty, przyjmując dane ze stdin (klawiatura) z pustym czasem kończącego się wejścia. Program przyjmuje maksymalnie trzy parametry: nazwę kolejki docelowej (wymagane), nazwę menedżera kolejek (opcjonalnie) oraz nazwę kolejki odpowiedzi (opcjonalnie). Jeśli nie zostanie podana

nazwa menedżera kolejek, zostanie użyty domyślny menedżer kolejek. Jeśli nie podano żadnej kolejki odpowiedzi, należy użyć komendy SYSTEM.SAMPLE.REPLY jest używana.

Poniżej przedstawiono przykład wywołania przykładowego programu C, określając kolejkę odpowiedzi, ale zezwalając na domyślne ustawienie menedżera kolejek:

```
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 do tworzenia pliku AMQSAMP4 mają nazwy utworzone w postaci wielkich liter.

Uruchamianie przykładu AMQ0REQ4

Program w języku COBOL tworzy komunikaty, akceptując dane z klawiatury. Aby uruchomić program, należy wywołać program i określić nazwę kolejki docelowej jako parametr. Program akceptuje wprowadzanie danych z klawiatury do buforu i tworzy komunikat żądania dla każdej linii tekstu. Program zostanie zatrzymany po wprowadzeniu pustego wiersza na klawiaturze.

Uruchamianie przykładu żądania przy użyciu wyzwalania

Jeśli próbka jest używana z wyzwalaniem i jednym z przykładowych programów Inquire, Set lub Echo, wiersz danych wejściowych musi być nazwą kolejki, do której ma być dostęp wyzwalany program.

ULW *Uruchamianie przykładu żądania przy użyciu wyzwalania w systemie UNIX, Linux, and Windows*

W systemie UNIX, Linux, and Windowsuruchom program monitor wyzwalacza RUNMQTRM w jednej sesji, a następnie uruchom program amqsreq w innej sesji.

Aby uruchomić przykłady za pomocą wyzwalania:

1. Uruchom program monitor wyzwalacza RUNMQTRM w jednej sesji (kolejka inicjowania SYSTEM.SAMPLE.TRIGGER jest dostępna do użycia).
2. Uruchom program amqsreq w innej sesji.
3. Upewnij się, że zdefiniowano docelową kolejkę serwera.

   Przykładowe kolejki, których można użyć jako docelowej kolejki serwera dla próbki żądania w celu umieszczenia komunikatów to:

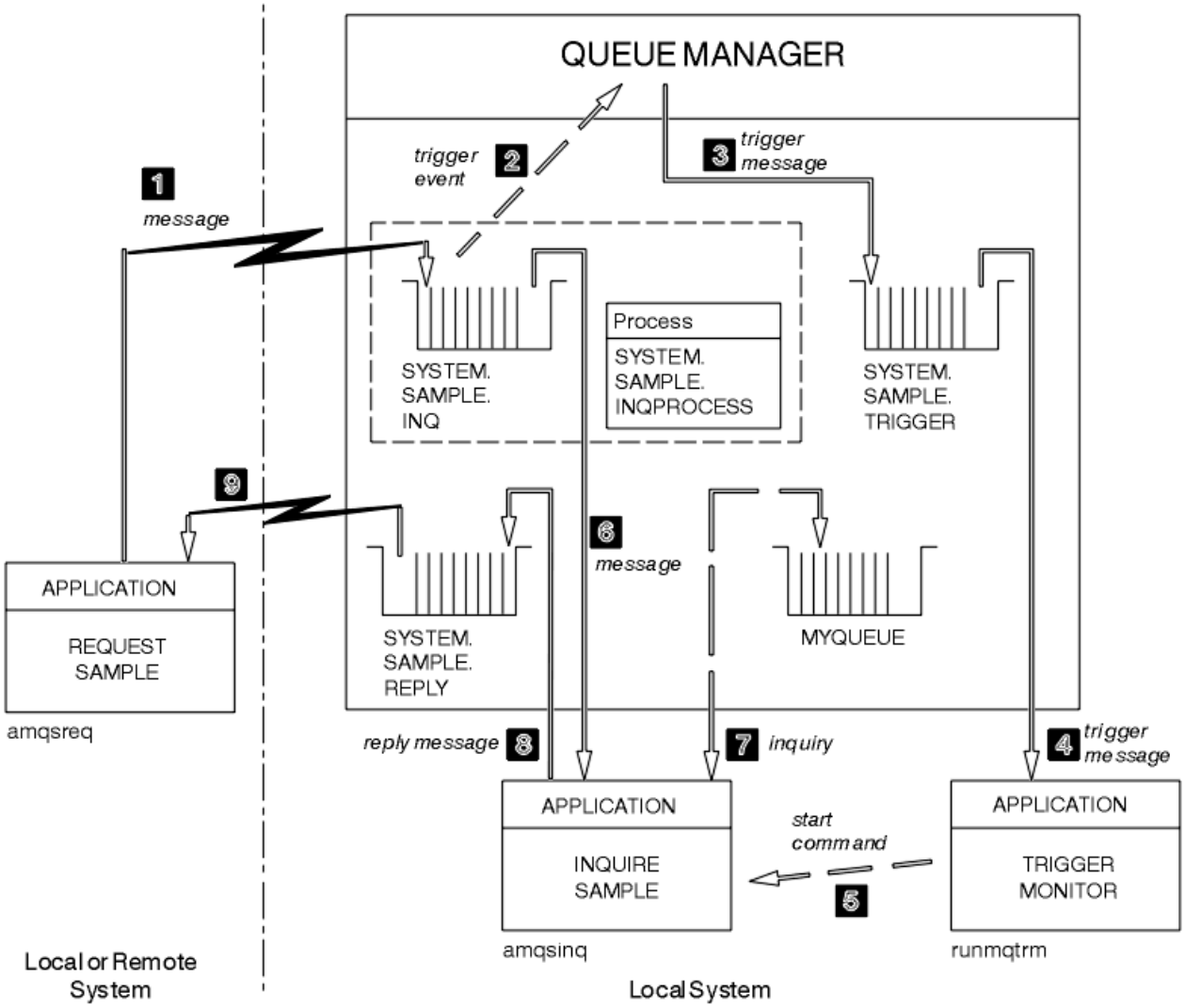
   - SYSTEM.SAMPLE.INQ -dla przykładowego programu Inquire
   - SYSTEM.SAMPLE.SET -dla przykładowego programu Set
   - SYSTEM.SAMPLE.ECHO -dla przykładowego programu Echo

   Kolejki te mają typ wyzwalacza FIRST, więc jeśli przed uruchomieniem przykładu żądania są już komunikaty w kolejkach, aplikacje serwera nie są wyzwalane przez wysyłane komunikaty.
4. Upewnij się, że zdefiniowano kolejkę dla przykładowego programu Inquire, Set lub Echo do użycia.

Oznacza to, że monitor wyzwalacza jest gotowy, gdy próbka żądania wysyła komunikat.

Uwaga: Przykładowe definicje procesów utworzone za pomocą komend RUNMQSC i amqscos0.tst powodują wyzwolenie próbek C. Zmień definicje procesów w pliku amqscos0.tst i użyj komendy RUNMQSC z tym zaktualizowanym plikiem w celu użycia wersji COBOL.

Rysunek 137 na stronie 1224 demonstruje, jak używać razem przykładów Request i Inquire.



Rysunek 137. Przykłady żądań i uzyskiwania informacji przy użyciu wyzwalania

W Rysunek 137 na stronie 1224 próbka żądania umieszcza komunikaty w docelowej kolejce serwera SYSTEM.SAMPLE.INQ, a próbka zapytania wysyła zapytanie do kolejki, MYQUEUE. Alternatywnie można użyć jednej z kolejek przykładowych zdefiniowanych po uruchomieniu komendy amqscos0.tstlub dowolnej innej zdefiniowanej kolejki, dla przykładu Inquire.

Uwaga: Numery w programie Rysunek 137 na stronie 1224 przedstawiają sekwencję zdarzeń.

Aby uruchomić przykłady żądań i uzyskiwania informacji przy użyciu wyzwalania:

1. Sprawdź, czy zdefiniowane są kolejki, które mają być używane. Uruchom komendę amqscos0.tst, aby zdefiniować przykładowe kolejki, a następnie zdefiniuj 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 zostać wyzwolone. Jeśli klient i serwer nie są uruchomione na tej samej platformie, wszystkie procesy uruchomione przez monitor wyzwalacza muszą definiować

ApplType, w przeciwnym razie serwer przyjmuje jego definicje domyślne (czyli typ aplikacji, która jest zwykle powiązana z maszyną serwera) i powoduje niepowodzenie.

Listę typów aplikacji można znaleźć w sekcji ApplType.

4. Wprowadź nazwę kolejki, która ma być używana przez próbkę Inquiry:

```
MYQUEUE
```

5. Wprowadź pusty wiersz (aby zakończyć działanie programu Request).
6. Następnie w przykładzie żądania zostanie wyświetlony komunikat zawierający dane programu Inquire uzyskanego z programu MYQUEUE.

W tym przypadku można użyć więcej niż jednej kolejki; w tym przypadku należy wprowadzić nazwy pozostałych kolejek w kroku “4” na stronie 1225.

Więcej informacji na temat wyzwalania zawiera sekcja “Uruchamianie aplikacji produktu IBM MQ przy użyciu wyzwalaczy” na stronie 958.

IBM i *Uruchamianie przykładu żądania przy użyciu wyzwalania w systemie IBM i*

W systemie IBM iuruchom przykładowy serwer wyzwalaczy, AMQSERV4, w jednym zadaniu, a następnie uruchom komendę AMQSREQ4 w innym zadaniu. Oznacza to, że serwer wyzwalacza jest gotowy, gdy przykładowy program żądania wysyła komunikat.

Uwaga:

1. Przykładowe definicje utworzone przez komendę AMQSAMP4 powodują wyzwolenie wersji 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 Change MQ Process (CHGMQMPRC)). W tym celu należy przeprowadzić edycję i uruchomić własną wersję produktu AMQSAMP4.
2. Kod źródłowy dla AMQSERV4 jest dostarczany tylko dla języka C. Jednak skompilowana wersja (której można użyć razem z próbkami języka COBOL) jest dostarczana w bibliotece QMQM.

Komunikaty żądań można umieścić w tych przykładowych kolejkach serwerów:

- SYSTEM.SAMPLE.ECHO (dla przykładowych programów Echo)
- SYSTEM.SAMPLE.INQ (dla przykładowych programów Inquire)
- SYSTEM.SAMPLE.SET (w przypadku zestawu przykładowych programów)

Wykres przepływu dla zmiennej SYSTEM.SAMPLE.ECHO jest wyświetlany w programie Rysunek 138 na stronie 1227. Za pomocą przykładowego pliku danych, którego komenda ma zostać wydana dla tego serwera, należy wykonać następujące komendy:

```
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(ECHO)')
```

Uwaga: Ta kolejka przykładowa ma typ wyzwalacza FIRST, więc jeśli przed uruchomieniem przykładu żądania są już komunikaty w kolejce, aplikacje serwera nie są wyzwalane przez wysyłane komunikaty.

Jeśli chcesz spróbować dalszych przykładów, możesz spróbować następujących wariantów:

- Użyj komendy AMQSTRG4 (lub jej równoważnego wiersza komend STRMQMTRM, aby uzyskać szczegółowe informacje na ten temat, patrz sekcja Uruchamianie monitora wyzwalacza MQ (STRMQMTRM)). zamiast AMQSERV4 do wprowadzenia zadania zamiast tego, ale potencjalne opóźnienia w składaniu zadań mogą sprawić, że będzie mniej łatwe w podążaniu za tym, co się dzieje.
- Uruchom plik SYSTEM.SAMPLE.INQUIRE i SYSTEM.SAMPLE.SET . Korzystając z przykładowego pliku danych, komendy służące do wydawania żądań programu C do tych serwerów są następujące:

```
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(INQ)')
CALL PGM(QMQMSAMP/AMQSREQ4) PARM('QMQMSAMP/AMQSDATA(SET)')
```

Te przykładowe kolejki również mają typ wyzwalacza FIRST.

Projekt przykładowego programu żądania
Program otwiera docelową kolejkę serwera, dzięki czemu może on umieszczać komunikaty. Korzysta on z wywołania MQOPEN z opcją MQOO_OUTPUT. Jeśli nie można otworzyć kolejki, w programie wyświetlany jest komunikat o błędzie zawierający kod przyczyny zwrócony przez wywołanie MQOPEN.

Następnie program otwiera kolejkę zwrotną o nazwie SYSTEM.SAMPLE.REPLY , aby można było uzyskać komunikaty odpowiedzi. W tym celu program korzysta z wywołania MQOPEN z opcją MQOO_INPUT_EXCLUSIVE. Jeśli nie można otworzyć kolejki, w programie wyświetlany jest komunikat o błędzie zawierający kod przyczyny zwrócony przez wywołanie MQOPEN.

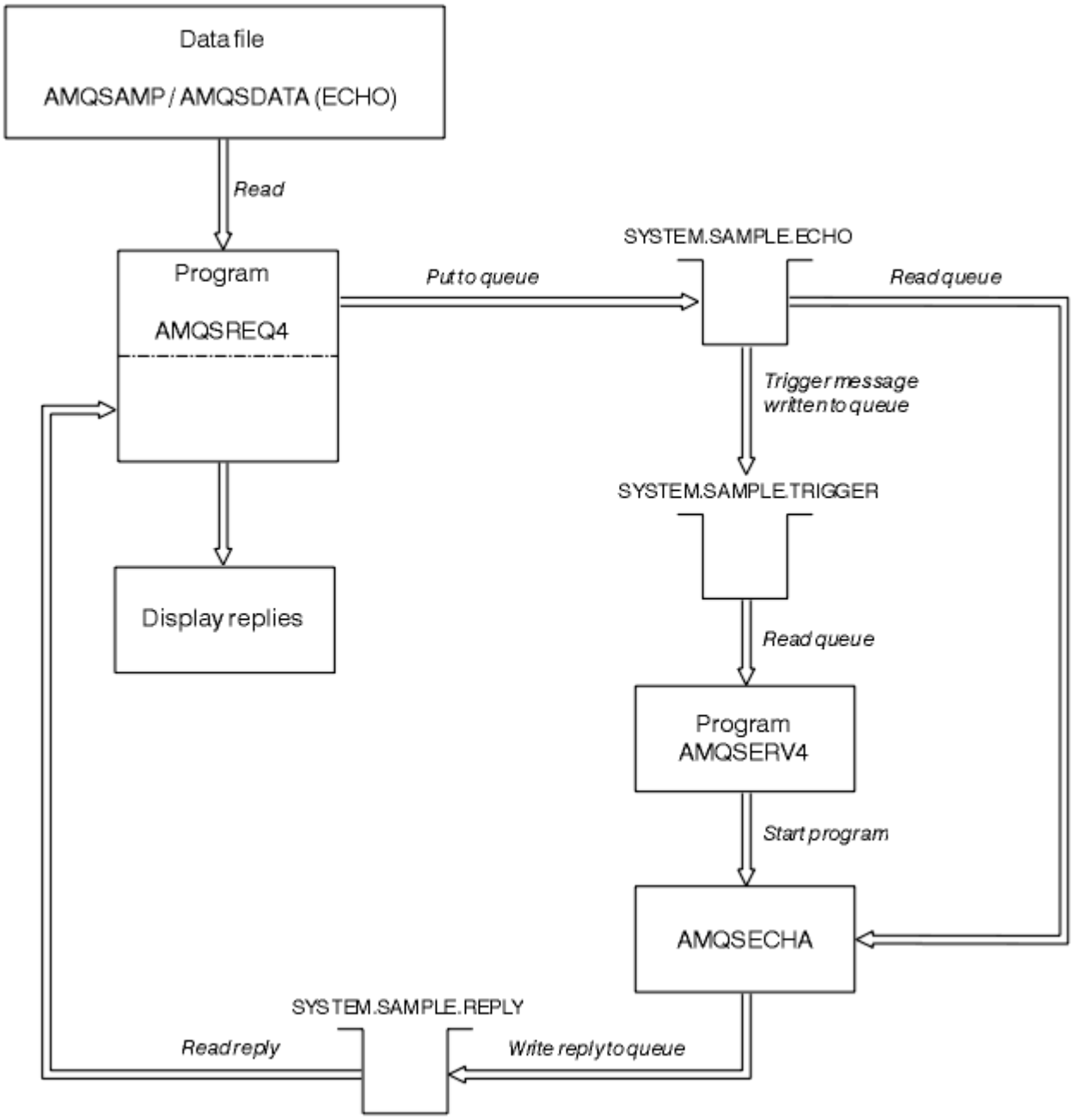
Dla każdego wiersza danych wejściowych program odczytuje tekst do buforu i korzysta z wywołania MQPUT w celu utworzenia komunikatu żądania zawierającego tekst tego wiersza. W tym wywołaniu program korzysta z opcji raportu MQRO_EXCEPTION_WITH_DATA, aby zażądać, aby wszystkie komunikaty raportu wysłane na temat komunikatu żądania zawierały pierwsze 100 bajtów danych komunikatu. Program będzie kontynuowany aż do momentu, gdy dojdzie do końca danych wejściowych lub wywołanie MQPUT nie powiedzie się.

Następnie program korzysta z wywołania MQGET w celu usunięcia komunikatów odpowiedzi z kolejki i wyświetla dane zawarte w odpowiedziach. Wywołanie MQGET korzysta z opcji MQGMO_WAIT, MQGMO_CONVERT i MQGMO_ACCEPT_OBCIĘTE. *WaitInterval* jest 5 minut w wersji COBOL i 1 minuta w wersji C, dla pierwszej odpowiedzi (w celu umożliwienia wyzwolenia aplikacji serwera) i 15 sekund na kolejne odpowiedzi. Program czeka na te okresy, jeśli w kolejce nie ma żadnego komunikatu. Jeśli przed upływem tego okresu nie zostanie wyświetlony żaden komunikat, wywołanie nie powiedzie się i zostanie zwrócony kod przyczyny MQRC_NO_MSG_AVAILABLE. Wywołanie korzysta również z opcji MQGMO_ACCEPT_TRUNCATED_MSG, dzięki czemu komunikaty dłuższe niż zadeklarowana wielkość buforu zostaną obcięte.

Program demonstruje, jak wyczyścić pola *MsgId* i *CorrelId* struktury MQMD po każdym wywołaniu MQGET, ponieważ wywołanie ustawia te pola na wartości zawarte w komunikacie, który pobiera. Usunięcie zaznaczenia tych pól oznacza, że kolejne wywołania MQGET pobierają komunikaty w kolejności, w jakiej komunikaty są przechowywane w kolejce.

Program będzie kontynuowany do czasu, aż wywołanie MQGET zwróci kod przyczyny MQRC_NO_MSG_AVAILABLE lub wywołanie MQGET nie powiedzie się. Jeśli wywołanie nie powiedzie się, w programie zostanie wyświetlony komunikat o błędzie zawierający kod przyczyny.

Następnie program zamyka docelową kolejkę serwera oraz kolejkę odpowiedzi przy użyciu wywołania MQCLOSE.



Rysunek 138. Przykładowy schemat blokowy programu IBM i Client/Server (Echo)

Przykładowe programy

W przykładowych programach ustaw operacje umieszczania w kolejce są wykonywane za pomocą wywołania MQSET w celu zmiany atrybutu **InhibitPut** kolejki. Ponadto należy zapoznać się z projektowaniem przykładowych programów Set.

Nazwy tych programów można znaleźć w sekcji “Funkcje demonstruje w przykładowych programach na Multiplatforms” na stronie 1168 .

Programy są przeznaczone do uruchamiania jako programy wyzwalane, więc ich jedynym wejściem jest struktura MQTMC2 (komunikat wyzwalany), która zawiera nazwę kolejki docelowej z atrybutami, które mają zostać zapytane. W wersji C używana jest również nazwa menedżera kolejek. W wersji COBOL używany jest domyślny menedżer kolejek.

Aby proces wyzwalania działał poprawnie, należy upewnić się, że program przykładowy Ustaw, który ma być używany, jest wyzwalany przez komunikaty przychodzące do kolejki SYSTEM.SAMPLE.SET. W tym celu

należy podać nazwę przykładowego programu Set, który ma być używany w polu *ApplicId* definicji procesu SYSTEM.SAMPLE.SETPROCESS. Kolejka przykładowa ma typ wyzwalacza FIRST; jeśli przed uruchomieniem przykładu żądania w kolejce istnieją już komunikaty, to próba ta nie jest wyzwalana przez wysyłane komunikaty.

Jeśli definicja została ustawiona poprawnie:

- ULW W przypadku systemów UNIX, Linux, and Windows należy uruchomić program **runmqtrm** w jednej sesji, a następnie uruchomić program amqsreq w innym.
- IBM i W przypadku produktu IBM iuruchom program AMQSERV4 w jednej sesji, a następnie uruchom program AMQSREQ4 w innej sesji. Można użyć komendy AMQSTRG4 zamiast AMQSERV4, ale ewentualne opóźnienia w składaniu zadań mogą spowodować, że śledzenie zdarzeń będzie mniej proste.

Za pomocą przykładowych programów żądania można wysyłać komunikaty żądań, z których każdy zawiera tylko nazwę kolejki, do kolejki SYSTEM.SAMPLE.SET. Dla każdego komunikatu żądania zestaw przykładowych programów wysyła komunikat odpowiedzi zawierający potwierdzenie, że operacje put zostały zahamowane w określonej kolejce. Odpowiedzi są wysyłane do kolejki odpowiedzi określonej w komunikacie żądania.

Projekt przykładowego programu Set

Program otwiera kolejkę nazwaną w strukturze komunikatu wyzwalacza, która została przekazana podczas jej uruchamiania. (Dla jasności zadzwonimy do tej *kolejki żądań*.) Program korzysta z wywołania MQOPEN, aby otworzyć tę kolejkę dla współużytkowanych danych wejściowych.

Program korzysta z wywołania MQGET w celu usunięcia komunikatów z tej kolejki. To wywołanie używa opcji MQGMO_ACCEPT_TRUNCATED_MSG i MQGMO_WAIT, z interwałem oczekiwania 5 sekund. Program testuje deskryptor każdego komunikatu, aby sprawdzić, czy jest to komunikat żądania. Jeśli tak nie jest, program usuwa komunikat i wyświetla komunikat ostrzegawczy.

Dla każdego komunikatu żądania usuniętego z kolejki żądań program odczytuje nazwę kolejki (którą wywołamy *kolejkę docelową*). zawarte w danych i otwierają tę kolejkę za pomocą wywołania MQOPEN z opcją MQOO_SET. Następnie program korzysta z wywołania MQSET w celu ustawienia wartości atrybutu **InhibitPut** kolejki docelowej na wartość MQQA_PUT_INHIBITED.

Jeśli wywołanie MQSET zakończy się pomyślnie, program korzysta z wywołania MQPUT1 w celu umieszczenia komunikatu odpowiedzi w kolejce odpowiedzi. Ten komunikat zawiera łańcuch PUT `inhibited`.

Jeśli wywołanie MQOPEN lub MQSET nie powiodło się, program korzysta z wywołania MQPUT1 w celu umieszczenia komunikatu `report` w kolejce odpowiedzi. W polu *Feedback* deskryptora komunikatu tego komunikatu raportu jest to kod przyczyny zwracany przez wywołanie MQOPEN lub MQSET, w zależności od tego, który błąd nie powiódł się.

Po wywołaniu komendy MQSET program zamknie kolejkę docelową przy użyciu wywołania MQCLOSE.

Jeśli w kolejce żądań nie pozostały żadne komunikaty, program zamknie tę kolejkę i rozłącza się z menedżerem kolejek.

Przykładowy program TLS

AMQSSLC to przykładowy program w języku C, który demonstruje sposób użycia struktur MQCNO i MQSCO w celu dostarczenia informacji o połączeniu klienta TLS w wywołaniu MQCONNX. Dzięki temu aplikacja MQI klienta udostępnia definicję kanału połączenia klienckiego i ustawienia TLS w czasie wykonywania bez tabeli definicji kanału klienta (CCDT).

Jeśli zostanie podana nazwa połączenia, program konstruuje definicję kanału połączenia klienta w strukturze MQCD.

Jeśli zostanie podana nazwa macierzysta pliku repozytorium kluczy, program konstruuje strukturę MQSCO. Jeśli adres URL programu odpowiadającego OCSP jest również podany, program konstruuje strukturę MQAIR rekordu informacji uwierzytelniającej.

Następnie program łączy się z menedżerem kolejek za pomocą komendy MQCONNX. Określa on i drukuje nazwę menedżera kolejek, z którym jest połączony.

Ten program jest przeznaczony do połączenia jako aplikacja kliencka MQI. Może być jednak dowiązany jako zwykła aplikacja MQI. Następnie, po prostu łączy się z lokalnym menedżerem kolejek i ignoruje informacje o połączeniu klienta.

Program AMQSSLC 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:

-k KeyReposStem
Nazwa macierzysta pliku repozytorium kluczy. Jest to pełna ścieżka do pliku bez przyrostka .kdb. Na przykład:

```
/home/user/client
C:\User\client
```

-s CipherSpec
Łańcuch TLS CipherSpec odpowiadający SSLCIPH w definicji kanału SVRCONN w menedżerze kolejek.

-f
Określa, że tylko algorytmy certyfikowane FIPS 140-2 muszą być używane.

-b VALUE1[,VALUE2...]
Określa, że tylko algorytmy zgodne z pakietem B muszą być używane. Ten parametr jest listą rozdzielaną przecinkami jednej lub kilku z następujących wartości: NONE,128_BIT,192_BIT. Wartości te mają takie samo znaczenie jak w przypadku zmiennej środowiskowej MQSUITEB, a równoważne ustawienie 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 każdą ze strategii sprawdzania poprawności certyfikatów obsługiwanych przez bibliotekę bezpiecznych gniazd i zaakceptuj łańcuch certyfikatów, jeśli dowolna z strategii uzna łańcuch certyfikatów za poprawny. To ustawienie może być używane w celu zapewnienia maksymalnej wstecznej zgodności 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 restrykcyjne sprawdzanie poprawności niż ustawienie ANY, ale odrzuca niektóre starsze certyfikaty cyfrowe.

Wartość domyślna to ANY.

-l CertLabel
Etykieta certyfikatu, która ma być używana dla bezpiecznego połączenia.

Uwaga: Należy podać wartość przy użyciu małych liter.

Parametr unieważnienia certyfikatu OCSP:

-o URL
Adres URL respondenta OCSP

Uruchamianie przykładowego programu TLS
Aby uruchomić przykładowy program TLS, należy najpierw skonfigurować środowisko TLS. Następnie można uruchomić przykład z poziomu wiersza komend, podając liczbę parametrów.

O tym zadaniu

Poniższe instrukcje uruchamiają program przykładowy przy użyciu certyfikatów osobistych. W zależności od komendy można na przykład użyć certyfikatów ośrodka CA i sprawdzić ich status za pomocą modułu odpowiadającego OCSP. Należy zapoznać się z instrukcjami w ramach przykładu.

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 produkcie UNIX, Linux, and Windows.
3. Utwórz repozytorium kluczy dla klienta. Wywołaj ją *clientkey.kdb*.
4. Utwórz certyfikat osobisty dla menedżera kolejek. Więcej informacji na ten temat zawiera sekcja Tworzenie samopodpisanego certyfikatu osobistego w systemie UNIX, 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 części publicznej certyfikatu samopodpisanego z repozytorium kluczy w systemie UNIX, Linux, and Windowsoraz Dodawanie certyfikatu ośrodka CA (lub publicznej części certyfikatu samopodpisanego) do repozytorium kluczy w systemach UNIX, Linux lub 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 Serwer-kanał połączenia .
9. Zdefiniuj i uruchom program nasłuchujący kanału w menedżerze kolejek. Więcej informacji na ten temat zawiera sekcja DEFINE LISTENER i START LISTENER.
10. Uruchom przykładowy program za pomocą następującej komendy:

    ```
    AMQSSSLC -m QM1 -c QM1SVRCONN -x localhost
    -k "C:\Program Files\IBM\MQ\clientkey" -s TLS_RSA_WITH_AES_128_CBC_SHA256
    -o http://dummy.OCSP.responder
    ```

Wyniki

Przykładowy program wykonuje następujące działania:

1. Łączy się z dowolnym określonym menedżerem kolejek lub z domyślnym menedżerem kolejek przy użyciu określonych opcji.
2. Służy do otwierania menedżera kolejek i uzyskiwania informacji o jego nazwie.
3. Zamyka menedżer kolejek.
4. Rozłącza się z menedżerem kolejek.

Jeśli przykładowy program zostanie uruchomiony pomyślnie, zostaną wyświetlone dane wyjściowe zbliżone do następującego przykładu:

```
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 przykładowy program napotka problem, zostanie wyświetlony odpowiedni komunikat o błędzie, na przykład w przypadku podania niepoprawnego adresu URL odpowiadającego OCSP, zostanie wyświetlony następujący komunikat:

```
MQCONNX ended with reason code 2553
```

Listę kodów przyczyny można znaleźć w sekcji Kody zakończenia i przyczyny API.

Przykładowe programy wyzwalające

Funkcja podana w przykładzie wyzwalającym jest podzbiorem tego udostępnionego w monitorze wyzwalacza w programie **runmqtrm** .

Nazwy tych programów można znaleźć w sekcji “Funkcje demonstruje w przykładowych programach na Multiplatforms” na stronie 1168 .

Wzór próby wyzwalającej

Program przykładowy wyzwalający otwiera kolejkę inicjującyprzy użyciu wywołania MQOPEN z opcją MQOO_INPUT_AS_Q_DEF. Pobiera on komunikaty z kolejki inicjuj przy użyciu wywołania MQGET z opcjami MQGMO_ACCEPT_TRUNCATED_MSG i MQGMO_WAIT, określając nieograniczony odstęp czasu oczekiwania. Program kasuje pola *MsgId* i *CorrelId* przed każdym wywołaniem MQGET w celu pobrania komunikatów w sekwencji.

Po pobraniu komunikatu z kolejki inicjuj program testuje komunikat, sprawdzając wielkość komunikatu, aby upewnić się, że jest to ta sama wielkość co struktura MQTM. Jeśli test zakończy się niepowodzeniem, program wyświetli ostrzeżenie.

W przypadku poprawnych komunikatów wyzwalacza próba wyzwalająca kopiuje dane z następujących pól: *ApplicId*, *EnvrData*, *Versio*n i *ApplType*. Ostatnie dwa z tych pól są numeryczne, dlatego program tworzy odpowiedniki znaków w celu użycia w strukturze MQTMC2 dla systemów IBM i UNIX, Linux, and Windows .

Próba wyzwalająca wysyła komendę uruchomienia do aplikacji określonej w polu *ApplicId* komunikatu wyzwalacza, a następnie przekazuje strukturę MQTMC2 lub MQTMC (wersja znakowa komunikatu wyzwalacza).

- ULW W systemach UNIX, Linux, and Windows pole *EnvData* jest używane jako rozszerzenie do wywołującego łańcucha komendy.
- IBM i W programie IBM ijest on używany jako parametry wprowadzania zadania, na przykład priorytet zadania lub opis zadania.

Na koniec program zamyka kolejkę inicjujący.

Zakończenie wyzwalania przykładowych programów w systemie IBM i

IBM i

Program monitora wyzwalacza może zostać zakończony przez opcję 2 (ENDRQS) sysrequest lub przez zahamowanie 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 wyzwalacza w tej kolejce należy wprowadzić następującą komendę:

```
CHGMQMQ QNAME('SYSTEM.SAMPLE.TRIGGER') GETENBL(*YES)
```

Uruchamianie przykładowych programów Triggering
Ten temat zawiera informacje na temat uruchamiania przykładowych programów Triggering.

Uruchamianie przykładów amqstrg0.c, amqstrg i amqstrgc

Program przyjmuje 2 parametry:

1. Nazwa kolejki inicjuj (niezbędnej)
2. Nazwa menedżera kolejek (opcjonalnie)

Jeśli menedżer kolejek nie jest określony, łączy się z domyślnym menedżerem kolejek. Przykładowa kolejka inicjujący zostanie zdefiniowana podczas uruchamiania komendy amqscos0.tst; , która jest nazwą kolejki SYSTEM.SAMPLE.TRIGGERi można go używać podczas uruchamiania tego programu.

Uwaga: Funkcja w tym przykładzie jest podzbiorem pełnej funkcji wyzwalającej, która jest dostarczana w programie runmqtrm.

Uruchamianie przykładu AMQSTRG4

IBM i

Jest to monitor wyzwalacza dla środowiska IBM i . Wprowadza ono jedno zadanie IBM i dla każdej aplikacji, która ma być uruchomiona. Oznacza to, że istnieje dodatkowe przetwarzanie powiązane z każdym komunikatem wyzwalacza.

Komenda AMQSTRG4 (w QCSRC) przyjmuje dwa parametry: nazwę kolejki inicjuj, która ma być używana, oraz nazwę menedżera kolejek (opcjonalnie). AMQSAMP4 (w QCLSRC) definiuje przykładową kolejkę inicjujową, SYSTEM.SAMPLE.TRIGGER, którego można użyć przy próbie próby programów przykładowych.

Korzystając z przykładowej kolejki wyzwalacza, komenda do wydania jest następująca:

```
CALL PGM(QMQM/AMQSTRG4) PARM('SYSTEM.SAMPLE.TRIGGER')
```

Alternatywnie można użyć równoważnego CL STRMQMTRM; aby uzyskać szczegółowe informacje, należy zapoznać się z informacjami w sekcji Uruchamianie monitora wyzwalacza MQ (STRMQMTRM).

Uruchamianie przykładu AMQSERV4

IBM i

Jest to serwer wyzwalacza dla środowiska IBM i . Dla każdego komunikatu wyzwalacza serwer uruchamia komendę uruchamiania w swoim własnym zadaniu, aby uruchomić określoną aplikację. Serwer wyzwalacza może wywoływać transakcje CICS .

Parametr AMQSERV4 przyjmuje dwa parametry: nazwę kolejki inicjuj, która ma być używana, oraz nazwę menedżera kolejek (opcjonalnie). AMQSAMP4 definiuje przykładową kolejkę inicjujową, SYSTEM.SAMPLE.TRIGGER, którego można użyć przy próbie próby programów przykładowych.

Za pomocą przykładowej kolejki wyzwalacza, której komenda ma zostać wydana, jest następująca komenda:

```
CALL PGM(QMQM/AMQSERV4) PARM('SYSTEM.SAMPLE.TRIGGER')
```

Projekt serwera wyzwalacza
Projekt serwera wyzwalacza jest podobny do projektu monitora wyzwalacza, z kilkoma wyjątkami.

Projekt serwera wyzwalającego jest podobny do projektu monitora wyzwalacza, z wyjątkiem tego, że serwer wyzwalacza:

- Umożliwia aplikacjom MQAT_CICS oraz MQAT_OS400 .
- IBM i Wywołuje aplikacje IBM i we własnym zadaniu (lub używa komendy STRCICSUSR do uruchamiania aplikacji CICS), a nie wysyłając zadania IBM i .
- W przypadku aplikacji CICS substytuuje *EnvData*, na przykład w celu określenia regionu CICS , z komunikatu wyzwalacza w komendzie STRCICSUSR.
- Otwiera kolejkę inicjują dla współużytkowanych danych wejściowych, dzięki czemu wiele serwerów wyzwalanych może być uruchomione w tym samym czasie.

Uwaga: Programy uruchomione przez program AMQSERV4 nie mogą korzystać z wywołania MQDISC, ponieważ powoduje to zatrzymanie serwera wyzwalacza. Jeśli programy uruchomione przez program AMQSERV4 korzystają z wywołania MQCONN, uzyskają kod przyczyny MQRC_ALREADY_CONNECTED.

Windows UNIX *Korzystanie z przykładów TUXEDO w systemach UNIX i Windows*

Dowiedz się więcej o programach umieszczania i pobierania próbek dla TUXEDO, a także w budowaniu środowiska 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 na więcej niż jedną linię. Nie wprowadzaj tego znaku. Każdą komendę należy wprowadzić jako pojedynczą linię.

Windows UNIX *Budowanie środowiska serwera*
Informacje na temat budowania środowiska serwera dla produktu IBM MQ dla różnych platform.

Zanim rozpoczniesz

Zakłada się, że użytkownik ma działające środowisko TUXEDO.

AIX *Budowanie środowiska serwera dla produktu AIX (wersja 32-bitowa)*
Jak zbudować środowisko serwera dla produktu IBM MQ for AIX (wersja 32-bitowa).

Procedura

1. Utwórz katalog (na przykład APPDIR), w którym środowisko serwera jest budowane i wykonuje wszystkie komendy znajdujące się w tym katalogu.
2. Wyeksportuj następujące zmienne środowiskowe, gdzie TUXDIR to katalog główny dla TUXEDO, a *MQ_INSTALLATION_PATH* reprezentuje katalog najwyższego 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 następujące 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. Edytuj plik ubbstxcx.cfg i dodaj w razie potrzeby 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. W tym pytaniu wpisz:

```
> crdl -z /APPDIR/TLOG1
```

7. Uruchom menedżer kolejek:

```
$ strmqm
```

8. Uruchom Tuxedo:

```
$ tmboot -y
```

Co dalej

Teraz można użyć programów do umieszczania komunikatów w kolejce i dogets w celu umieszczenia komunikatów w kolejce i pobrania ich z kolejki.

AIX *Budowanie środowiska serwera dla produktu AIX (64-bitowego)*
Jak zbudować środowisko serwera dla produktu IBM MQ for AIX (64-bitowego).

Procedura

1. Utwórz katalog (na przykład APPDIR), w którym środowisko serwera jest budowane i wykonuje wszystkie komendy znajdujące się w tym katalogu.
2. Wyeksportuj następujące zmienne środowiskowe, gdzie TUXDIR reprezentuje katalog główny dla TUXEDO, a *MQ_INSTALLATION_PATH* reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest 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 następujące 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. Edytuj plik ubbstxcx.cfg i dodaj w razie potrzeby 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. W tym pytaniu wpisz:

```
> crdl -z /APPDIR/TLOG1
```

7. Uruchom menedżer kolejek:

```
$ strmqm
```

8. Uruchom Tuxedo:

```
$ tmboot -y
```

Co dalej

Teraz można użyć programów do umieszczania komunikatów w kolejce i dogets w celu umieszczenia komunikatów w kolejce i pobrania ich z kolejki.

Solaris *Budowanie środowiska serwera dla produktu Solaris (wersja 32-bitowa)*

Jak zbudować środowisko serwera dla produktu IBM MQ for Solaris (wersja 32-bitowa).

O tym zadaniu

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Procedura

1. Utwórz katalog (na przykład APPDIR), w którym środowisko serwera jest budowane i wykonuje wszystkie komendy znajdujące się w tym katalogu.
2. Wyeksportuj następujące zmienne środowiskowe, gdzie TUXDIR to katalog główny dla TUXEDO:

```
$ export CFLAGS="-I /APPDIR"
$ export FIELDTBLS=amqstxvx.flds
$ export VIEWFILES=amqstxvx.V
$ export SHLIB_PATH=$TUXDIR/lib:MQ_INSTALLATION_PATH/lib:/lib
$ export LD_LIBRARY_PATH=$TUXDIR/lib:MQ_INSTALLATION_PATH/lib:/lib
```

3. Dodaj następujące informacje do pliku TUXEDO udataobj/RM (RM must include *MQ_INSTALLATION_PATH*/lib/libmqmcs and *MQ_INSTALLATION_PATH*/lib/libmqmzse).

```
MQSeries_XA_RMI:MQRMIXASwitchDynamic: \
MQ_INSTALLATION_PATH/lib/libmqmxa.a MQ_INSTALLATION_PATH/lib/libmqm.so \
/opt/tuxedo/lib/libtux.a MQ_INSTALLATION_PATH/lib/libmqmcs.so \
MQ_INSTALLATION_PATH/lib/libmqmzse.so
```

4. Uruchom następujące komendy:

```
$ mkfldhdr    amqstxvx.flds
$ viewc       amqstxvx.v
$ buildtms    -o MQXA -r MQSERIES_XA_RMI
$ buildserver -o MQSERV1 -f amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so \
       -r MQSERIES_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
       -l -ldl
$ buildserver -o MQSERV2 -f amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so \
       -r MQSERIES_XA_RMI -s MPUT2:MPUT \
       -s MGET2:MGET \
       -v -bshm
       -l -ldl
$ buildclient -o doputs -f amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so \
       -f MQ_INSTALLATION_PATH/lib/libmqmzse.co \
       -f MQ_INSTALLATION_PATH/lib/libmqmcs.so
$ buildclient -o dogets -f amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib/libmqm.so
       -f MQ_INSTALLATION_PATH/lib/libmqmzse.co \
       -f MQ_INSTALLATION_PATH/lib/libmqmcs.so
```

5. Edytuj ubbstxcx.cfg i dodaj szczegóły dotyczące nazwy komputera, katalogów roboczych i menedżera kolejek, jeśli jest to konieczne:

```
$ tmloadcf  -y ubbstxcx.cfg
```

6. Utwórz TLOGDEVICE:

```
$tmadmin -c
```

   Zostanie wyświetlona zachęta. W tym pytaniu wpisz:

```
> crdl -z /APPDIR/TLOG1
```

7. Uruchom menedżer kolejek:

```
$ strmqm
```

8. Uruchom Tuxedo:

```
$ tmboot -y
```

Co dalej

Teraz można użyć programów do umieszczania komunikatów w kolejce i dogets w celu umieszczenia komunikatów w kolejce i pobrania ich z kolejki.

Solaris *Budowanie środowiska serwera dla produktu Solaris (64-bitowego)*
Jak zbudować środowisko serwera dla produktu IBM MQ for Solaris (64-bitowego).

O tym zadaniu

MQ_INSTALLATION_PATH reprezentuje katalog najwyższego poziomu, w którym zainstalowany jest produkt IBM MQ .

Procedura

1. Utwórz katalog (na przykład APPDIR), w którym środowisko serwera jest budowane i wykonuje wszystkie komendy znajdujące się w tym katalogu.
2. Wyeksportuj następujące zmienne środowiskowe, gdzie TUXDIR to katalog główny dla TUXEDO:

```
$ export CFLAGS="-I /APPDIR"
$ export FIELDTBLS=amqstxvx.flds
$ export VIEWFILES=amqstxvx.V
$ export SHLIB_PATH=$TUXDIR/lib: MQ_INSTALLATION_PATH/lib:/lib64
$ export LD_LIBRARY_PATH=$TUXDIR/lib64: MQ_INSTALLATION_PATH/lib64:/lib64
```

3. Dodaj następujące informacje do pliku TUXEDO udataobj/RM (RM must include *MQ_INSTALLATION_PATH*/lib/libmqmcs and *MQ_INSTALLATION_PATH*/lib/libmqmzse).

```
MQSERIES_XA_RMI:MQRMIXASwitchDynamic: \
MQ_INSTALLATION_PATH/lib64/libmqmxa64.a MQ_INSTALLATION_PATH/lib64/libmqm.so \
/opt/tuxedo/lib64/libtux.a MQ_INSTALLATION_PATH/lib64/libmqmcs.so \
MQ_INSTALLATION_PATH/lib64/libmqmzse.so
```

4. Uruchom następujące komendy:

```
$ mkfldhdr    amqstxvx.flds
$ viewc       amqstxvx.v
$ buildtms    -o MQXA -r MQSERIES_XA_RMI
$ buildserver -o MQSERV1 -f amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so \
       -r MQSERIES_XA_RMI -s MPUT1:MPUT \
       -s MGET1:MGET \
       -v -bshm
       -l -ldl
$ buildserver -o MQSERV2 -f amqstxsx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so \
       -r MQSERIES_XA_RMI -s MPUT2:MPUT \
       -s MGET2:MGET \
       -v -bshm
       -l -ldl
$ buildclient -o doputs -f amqstxpx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so \
       -f MQ_INSTALLATION_PATH/lib64/libmqmzse.co \
       -f MQ_INSTALLATION_PATH/lib64/libmqmcs.so
$ buildclient -o dogets -f amqstxgx.c \
       -f MQ_INSTALLATION_PATH/lib64/libmqm.so
       -f MQ_INSTALLATION_PATH/lib64/libmqmzse.co \
       -f MQ_INSTALLATION_PATH/lib64/libmqmcs.so
```

5. Edytuj ubbstxcx.cfg i dodaj szczegóły dotyczące nazwy komputera, katalogów roboczych i menedżera kolejek, jeśli jest to konieczne:

```
$ tmloadcf  -y ubbstxcx.cfg
```

6. Utwórz TLOGDEVICE:

```
$tmadmin -c
```

   Zostanie wyświetlona zachęta. W tym pytaniu wpisz:

```
> crdl -z /APPDIR/TLOG1
```

7. Uruchom menedżer kolejek:

```
$ strmqm
```

8. Uruchom Tuxedo:

```
$ tmboot -y
```

Co dalej

Teraz można użyć programów do umieszczania komunikatów w kolejce i dogets w celu umieszczenia komunikatów w kolejce i pobrania ich z kolejki.

Windows *Budowanie środowiska serwera dla produktu Windows (wersja 32-bitowa)*
Budowanie środowiska serwera dla produktu IBM MQ for Windows (wersja 32-bitowa).

O tym zadaniu

Uwaga: Zmień pola oznaczone jako *VARIABLES* (zmienne) w następujące, do ścieżek katalogów:

Tabela 171. Pola, które mają zostać zmienione na ścieżki katalogów

Pole	**Ścieżka do katalogu**
MQMDIR	Ścieżka do katalogu określona podczas instalowania produktu IBM MQ , na przykład `g:\Program Files\IBM\MQ`.
KATALOG_TUXDIR	Ścieżka do 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 139. Przykład pliku ubbstxcn.cfg dla produktu IBM MQ for Windows

Uwaga: Zmień nazwę komputera *MachineName*i ścieżki do katalogu, aby dopasować ją do instalacji. Zmień także nazwę menedżera kolejek *MYQUEUEMANAGER* na nazwę menedżera kolejek, z którym ma zostać nawiązane połączenie.

Przykładowy plik ubbconfig dla IBM MQ for Windows znajduje się na liście Rysunek 139 na stronie 1239. Jest on dostarczany jako ubbstxcn.cfg w katalogu przykładów produktu IBM MQ .

Przykładowy plik makefile (patrz Rysunek 140 na stronie 1240) dostarczany dla IBM MQ for Windows nosi nazwę ubbstxmn.maki znajduje się w katalogu przykładów produktu 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 140. Przykładowy plik makefile TUXEDO dla IBM MQ for Windows

Aby zbudować środowisko serwera i przykłady, wykonaj następujące kroki.

Procedura

1. Utwórz katalog aplikacji, w którym ma zostać utworzona przykładowa aplikacja, na przykład:

   ```
   f:\tuxedo\apps\mqapp
   ```

2. Skopiuj następujące przykładowe pliki z przykładowego katalogu IBM MQ do katalogu aplikacji:
   - `amqstxmn.mak`
   - `amqstxen.env`
   - `ubbstxcn.cfg`
3. Zmodyfikuj każdy z tych plików, aby ustawić nazwy katalogów i ścieżki katalogów używane podczas instalacji.
4. Edytuj `ubbstxcn.cfg` (patrz Rysunek 139 na stronie 1239), aby dodać szczegółowe informacje o nazwie komputera i menedżerze 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
   ```

   Nowa pozycja musi być jednym z wierszy 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.

   Aby to zrobić, wywołaj komendę `tmadmin -c` i wprowadź następującą komendę:

   ```
   crdl -z APPDIR\TLOG
   ```

8. Ustaw bieżący katalog na *APPDIR*, a następnie wywołaj przykładowy plik makefile `amqstxmn.mak` jako zewnętrzny plik makefile projektu. Na przykład przy użyciu programu Microsoft Visual C++ wprowadź następującą komendę:

   ```
   msvc amqstxmn.mak
   ```

   Wybierz opcję **build** (kompilacja), aby zbudować wszystkie przykładowe programy.

Windows *Budowanie środowiska serwera dla produktu Windows (64-bitowego)*

Jak zbudować środowisko serwera dla produktu IBM MQ for Windows (64-bitowego).

O tym zadaniu

Uwaga: Zmień pola oznaczone jako *VARIABLES* (zmienne) w następujące, do ścieżek katalogów:

Tabela 172. Pola, które mają zostać zmienione na ścieżki katalogów

Pole	**Ścieżka do katalogu**
MQMDIR	Ścieżka do katalogu określona podczas instalowania produktu IBM MQ , na przykład `g:\Program Files\IBM\MQ`.
KATALOG_TUXDIR	Ścieżka do 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 141. Przykład pliku ubbstxcn.cfg dla produktu IBM MQ for Windows

Uwaga: Zmień nazwę komputera *MachineName*i ścieżki do katalogu, aby dopasować ją do instalacji. Zmień także nazwę menedżera kolejek *MYQUEUEMANAGER* na nazwę menedżera kolejek, z którym ma zostać nawiązane połączenie.

Przykładowy plik ubbconfig dla IBM MQ for Windows znajduje się na liście Rysunek 141 na stronie 1242. Jest on dostarczany jako `ubbstxcn.cfg` w katalogu przykładów produktu IBM MQ .

Przykładowy plik makefile (patrz Rysunek 142 na stronie 1243) dostarczane dla IBM MQ for Windows nosi nazwę `ubbstxmn.mak`i jest przechowywane w katalogu przykładów produktu 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 142. Przykładowy plik makefile TUXEDO dla IBM MQ for Windows

Aby zbudować środowisko serwera i przykłady, wykonaj następujące kroki.

Procedura

1. Utwórz katalog aplikacji, w którym ma zostać utworzona przykładowa aplikacja, na przykład:

   ```
   f:\tuxedo\apps\mqapp
   ```

2. Skopiuj następujące przykładowe pliki z przykładowego katalogu IBM MQ do katalogu aplikacji:
   - `amqstxmn.mak`
   - `amqstxen.env`
   - `ubbstxcn.cfg`
3. Zmodyfikuj każdy z tych plików, aby ustawić nazwy katalogów i ścieżki katalogów używane podczas instalacji.
4. Edytuj `ubbstxcn.cfg` (patrz Rysunek 141 na stronie 1242) 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
   ```

   Nowa pozycja musi być jednym z wierszy 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. Aby to zrobić, wywołaj komendę `tmadmin -c`i wprowadź komendę:

   ```
   crdl -z APPDIR\TLOG
   ```

8. Ustaw bieżący katalog na *APPDIR*, a następnie wywołaj przykładowy plik makefile `amqstxmn.mak` jako zewnętrzny plik makefile projektu. Na przykład przy użyciu programu Microsoft Visual C++ wprowadź następującą komendę:

   ```
   msvc amqstxmn.mak
   ```

   Wybierz opcję **build** (kompilacja), aby zbudować wszystkie przykładowe programy.

Przykładowy program serwera dla TUXEDO

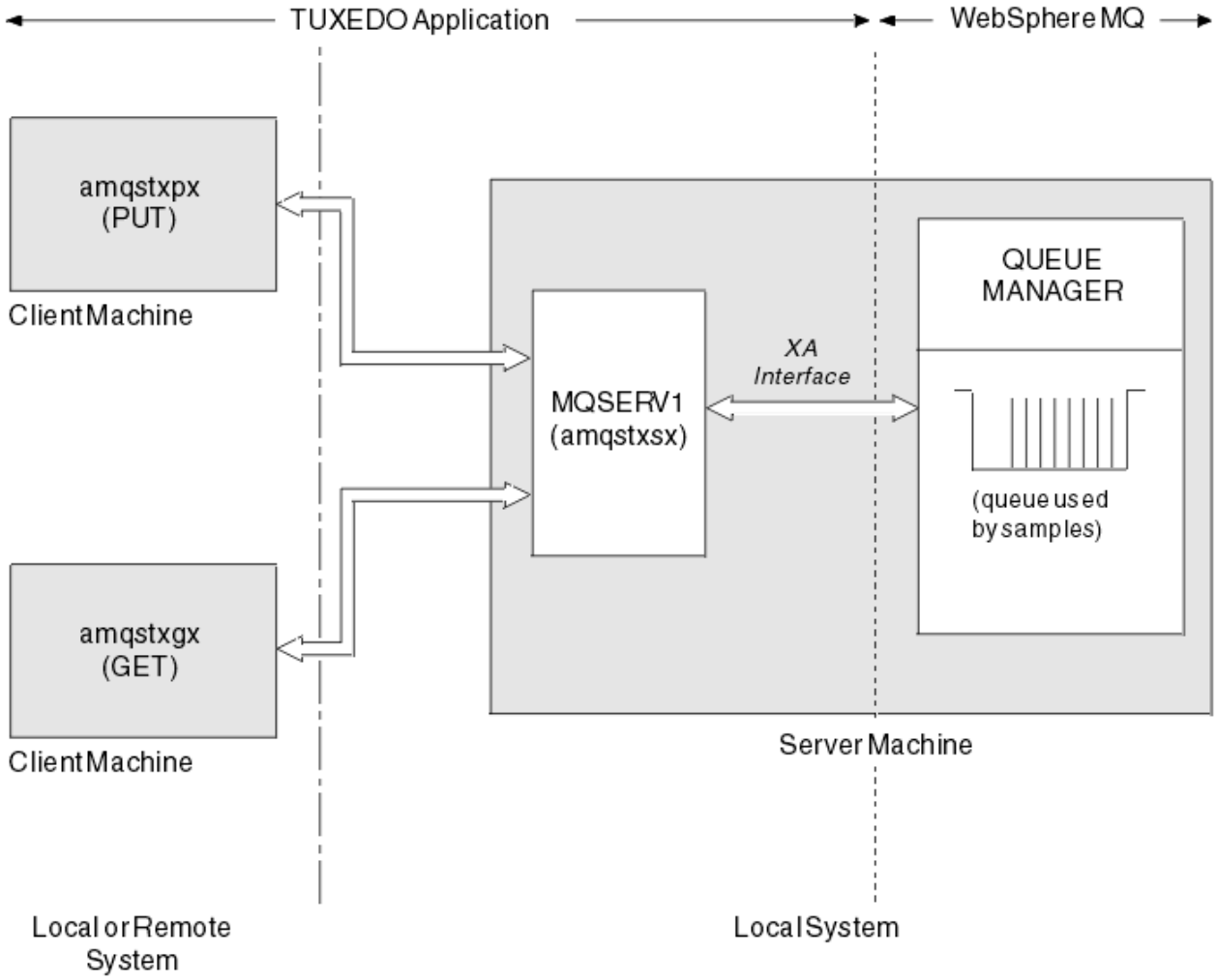
Przykładowy program serwera (amqstxsx) jest przeznaczony do uruchamiania z przykładowymi programami Put (amqstxpx.c) i Get (amqstxgx.c). Przykładowy program serwera jest uruchamiany automatycznie po uruchomieniu TUXEDO.

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 korzysta z komendy MQPUT1 w punkcie synchronizacji w celu umieszczenia komunikatu w jednostce pracy sterowanej przez TUXEDO. Pobiera ona parametry QName i tekst komunikatu, które są dostarczane przez przykład PUT.
- Usługa MGET1 zostanie otwarta i zamyka kolejkę za każdym razem, gdy zostanie wyświetlony komunikat. Pobiera ona parametry QName i tekst komunikatu, które są dostarczane przez przykład GET.

Wszystkie komunikaty o błędach, kody przyczyny i komunikaty o statusie są zapisywane w pliku dziennika TUXEDO.



Rysunek 143. Jak próbki TUXEDO współpracują ze sobą

Windows UNIX *Umieść przykładowy program dla TUXEDO*

Ten przykład umożliwia wielokrotne umieszczanie komunikatu w kolejce w partiach, demonstrując za pomocą TUXEDO funkcję syncwskazujący jako menedżera zasobów.

Przykładowy program serwera `amqstxsx` musi być uruchomiony, aby próba umieszczenia przykładu powiodła się; przykładowy program serwera łączy się 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 każda zawiera pięć komunikatów. Jeśli wystąpią jakiekolwiek problemy, wycofuje ona zadanie wsadowe komunikatów, w przeciwnym razie je zatwierdzije.

Wszystkie komunikaty o błędach są zapisywane w pliku dziennika TUXEDO i do standardowego wyjścia błędów. Wszelkie kody przyczyny są zapisywane w stderr.

Windows UNIX *Pobierz przykład dla TUXEDO*

Ten przykład umożliwia pobieranie komunikatów z kolejki w zadaniach wsadowych.

Przykładowy program serwera `amqstxsx` musi być uruchomiony, aby próba pobrania powiodła się; przykładowy program serwera łączy się z menedżerem kolejek i korzysta z 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 wyłączenie 24 komunikatów z kolejki o nazwie `myqueue`, w sześciu partiach, z których każda zawiera cztery komunikaty. W przypadku uruchomienia tego przykładu po umieszczonym przykładzie, który umieszcza 30 komunikatów w systemie `myqueue`, w systemie `myqueue`będzie mieć tylko sześć komunikatów. Liczba zadań wsadowych i wielkość zadania wsadowego mogą być różne między umieszczanie komunikatów i ich pobieranie.

Wszystkie komunikaty o błędach są zapisywane w pliku dziennika TUXEDO i do standardowego wyjścia błędów. Wszelkie kody przyczyny są zapisywane w stderr.

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 znajduje się w dwóch formatach: obiektowym i źródłowym.

Kod obiektu

Plik z kodem obiektu nosi nazwę `amqrspin.dll`. Zarówno dla klienta, jak i dla serwera, jest on instalowany jako standardowy element programu 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 standardowe wyjście użytkownika. Można uruchomić dostarczone wyjście kanału zabezpieczeń i użyć usług uwierzytelniania w definicji kanału.

W tym celu należy określić jedną z następujących wartości:

```
SCYEXIT('amqrspin(SCY_KERBEROS)')
SCYEXIT('amqrspin(SCY_NTLM)')
```

Aby udostępnić obsługę dla kanału zastrzeżonego, należy określić następujące informacje na kanale SVRCONN:

```
SCYDATA('remote_principal_name')
```

gdzie *nazwa_zdalnej_principal_name* ma postać *DOMAIN**użytkownik*. Bezpieczny kanał jest ustanawiany tylko wtedy, gdy nazwa zdalnego użytkownika jest zgodna z nazwą *nazwa_zdalnej_principalna_nazwa*.

Aby użyć dostarczonych programów obsługi wyjścia kanału między systemami, które działają w obrębie domeny zabezpieczeń Kerberos , należy utworzyć **servicePrincipalName** dla menedżera kolejek.

Kod źródłowy

Plik kodu źródłowego wyjścia ma nazwę `amqsspin.c`. Jest on dostępny w produkcie `C:\Program Files\IBM\MQ\Tools\c\Samples`.

W przypadku zmodyfikowania kodu źródłowego należy ponownie skompilować zmodyfikowane źródło.

Należy je skompilować i połączyć w taki sam sposób, jak inne wyjście kanału dla odpowiedniej platformy, z tą różnicą, że nagłówki SSPI muszą być dostępne w czasie kompilacji, a biblioteki bezpieczeństwa SSPI, wraz z dowolnymi rekomendowanymi powiązanymi bibliotekami, muszą być dostępne w czasie połączenia.

Przed wykonaniem poniższej komendy należy upewnić się, że w ścieżce znajdują się `cl.exe`oraz 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 żadnego przepisu dotyczącego śledzenia lub obsługi błędów. W przypadku zmodyfikowania i użycia kodu źródłowego należy dodać własne procedury śledzenia i obsługi błędów.

Uruchamianie przykładów przy użyciu kolejek zdalnych

Istnieje możliwość zademonstrowania zdalnego kolejkowania przez uruchomienie przykładów w połączonych menedżerach kolejek.

Program amqscos0.tst udostępnia lokalną definicję kolejki zdalnej (SYSTEM.SAMPLE.REMOTE), w którym używany jest zdalny menedżer kolejek o nazwie OTHER. Aby użyć tej definicji przykładu, należy zmienić wartość inną na nazwę drugiego menedżera kolejek, który ma być używany. Należy również skonfigurować kanał komunikatów między dwoma menedżerami kolejek. Aby uzyskać informacje na temat sposobu wykonania tej czynności, należy zapoznać się z informacjami w sekcji Definiowanie kanałów.

Przykładowe programy żądania umieją umieścić własną nazwę lokalnego menedżera kolejek w polu *ReplyToQMgr* wysyłanych przez nich komunikatów. Przykłady Inquire i Set wysyłają komunikaty odpowiedzi do kolejki i menedżera kolejek komunikatów o nazwie w polach *ReplyToQ* i *ReplyToQMgr* komunikatów żądania, które przetwarzali.

Przykładowy program monitorowania kolejki klastra (AMQSCLM)

Ten przykład korzysta z wbudowanych funkcji równoważenia obciążenia klastra produktu IBM MQ w celu kierowania komunikatów do instancji kolejek, które używają przyłączonych aplikacji. Ten automatyczny kierunek zapobiega gromadzeniu komunikatów w instancji kolejki klastra, do której nie jest przyłączona aplikacja konsumująca.

Przegląd

Istnieje możliwość skonfigurowania klastra, który ma więcej niż jedną definicję dla tej samej kolejki w różnych menedżerach kolejek. Ta konfiguracja zapewnia korzyści ze zwiększonej dostępności i równoważenia obciążenia. Jednak nie ma możliwości wbudowanej w produkt IBM MQ w celu dynamicznego modyfikowania dystrybucji komunikatów w klastrze w oparciu o stan dołączonych aplikacji. Z tego powodu aplikacja konsumująca zawsze musi być przyłączona do każdej instancji kolejki, aby zapewnić, że komunikaty będą przetwarzane.

Przykładowy program monitorujący kolejkę klastra monitoruje stan dołączonych aplikacji. Program dynamicznie dopasowuje wbudowaną konfigurację równoważenia obciążenia w celu kierowania komunikatów do instancji kolejki klastrowej z dołączonymi aplikacjami wykorzystudzonymi. W pewnych sytuacjach program ten może być używany do odprężania potrzeb aplikacji konsumowanych, aby zawsze były połączone z każdą instancją kolejki. Powoduje również ponowne wysyłanie komunikatów, które zostały umieszczone w kolejce w instancji kolejki, w której nie są przyłączone żadne aplikacje korzystające z aplikacji. Ponowne wysyłanie komunikatów umożliwia skierowanie komunikatów wokół aplikacji konsumowania, która jest tymczasowo zamknięta.

Program jest przeznaczony do użycia w przypadku, gdy aplikacje korzystające z pracy są aplikacjami długotrwałych, a nie często dołączaniem i odłączaniem aplikacji.

Przykładowy program monitorujący kolejkę klastra jest skompilowanym programem wykonywalnym przykładowego pliku C `amqsclma.c`.

Więcej informacji na temat klastrów i obciążenia można znaleźć w sekcji Używanie klastrów do zarządzania obciążeniem .

AMQSCLM: Projektowanie i planowanie korzystania z przykładu
Informacje na temat sposobu działania przykładowego programu monitorującego kolejkę klastra, punktów, które należy wziąć pod uwagę podczas konfigurowania systemu dla przykładowego programu, który ma być uruchomiony, oraz modyfikacji, jakie można wprowadzić w przykładowym kodzie źródłowym.

Projektowanie

Przykładowy program monitorujący kolejkę klastra monitoruje lokalne kolejki klastrowe, które używają przyłączonych aplikacji. Program monitoruje kolejki określone przez użytkownika. Nazwa kolejki może być konkretna, na przykład APP.TEST01, lub nazwa ogólna. Nazwy ogólne muszą być w formacie zgodnym z PCF (Programmable Command Format). Przykłady nazw ogólnych to APP.TEST*lub APP*.

Każdy menedżer kolejek w klastrze, który jest właścicielem instancji kolejki lokalnej, która ma być monitorowana, wymaga, aby program przykładowy monitorowania kolejki klastra był połączony z tym programem.

Dynamiczne kierowanie komunikatów

Przykładowy program monitorujący kolejkę klastra używa wartości **IPPROCS** (open for input process count) kolejki w celu określenia, czy ta kolejka ma konsumentów. Wartość większa od 0 wskazuje, że kolejka ma przyłączoną co najmniej jedną aplikację konsumująca. Takie kolejki są aktywne. Wartość 0 oznacza, że w kolejce nie są przyłączone programy korzystające z pamięci. Takie kolejki są nieaktywne.

W przypadku kolejki klastrowej z wieloma instancjami w klastrze produkt IBM MQ używa właściwości priorytetu obciążenia klastra **CLWLPRTY** każdej instancji kolejki w celu określenia instancji, do których mają być wysyłane komunikaty. Program IBM MQ wysyła komunikaty do dostępnych instancji kolejki o największej wartości **CLWLPRTY** .

Przykładowy program monitorujący kolejkę klastra aktywuje kolejkę klastra przez ustawienie lokalnej wartości **CLWLPRTY** na 1. Program dezaktywuje kolejkę klastra przez ustawienie jej wartości **CLWLPRTY** na 0.

Technologia klastrowa produktu IBM MQ propaguje zaktualizowaną właściwość **CLWLPRTY** kolejki klastrowej do wszystkich odpowiednich menedżerów kolejek w klastrze. Na przykład składnia

- Menedżer kolejek z podłączonym aplikacją, która 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 największej wartości **CLWLPRTY** w obrębie klastra.

Przesyłanie komunikatów w kolejce

Dynamiczna modyfikacja wartości **CLWLPRTY** wpływa na kierowanie nowych komunikatów. Ta dynamiczna modyfikacja nie ma wpływu na komunikaty znajdujące się już w kolejce w instancji kolejki bez przyłączonych konsumentów lub komunikaty, które przeszły mechanizm równoważenia obciążenia przed propagowaniem zmodyfikowanej wartości produktu **CLWLPRTY** przez klaster. W wyniku tego komunikaty pozostają w żadnej nieaktywnej kolejce i nie są przetwarzane przez aplikację konsumująca. Aby rozwiązać ten problem, przykładowy program monitorujący kolejkę klastra jest w stanie pobrać komunikaty z lokalnej kolejki bez konsumentów i wysłać te komunikaty do zdalnych instancji tej samej kolejki, w której są przyłączone konsumenty.

Przykładowy program monitorujący kolejkę klastra przesyła komunikaty z nieaktywnej kolejki lokalnej do jednej lub większej liczby aktywnych kolejek zdalnych, pobierając komunikaty (za pomocą **MQGET**). i umieszczanie komunikatów (za pomocą programu **MQPUT**) do tej samej kolejki klastrowej. Ten transfer powoduje, że zarządzanie obciążeniem klastra IBM MQ wybiera inną instancję docelową w oparciu o wartość CLWLPRTY większą niż wartość instancji kolejki lokalnej. Trwałość komunikatu i kontekst są zachowywane podczas przesyłania komunikatów. Kolejność komunikatów i wszelkie opcje powiązania nie są zachowywane.

Planowanie

Przykładowy program monitorujący kolejkę klastra modyfikuje konfigurację klastra, gdy istnieje zmiana w połączeniach aplikacji konsumujących. Modyfikacje są przesyłane z menedżerów kolejek, w których przykładowy program monitorujący kolejkę klastra monitoruje kolejki, do pełnych menedżerów kolejek repozytorium w klastrze. Menedżery kolejek pełnego repozytorium przetwarają aktualizacje konfiguracji i ponownie je wznawiają we wszystkich odpowiednich menedżerach kolejek w klastrze. Odpowiednie menedżery kolejek obejmują te menedżery kolejek, które są właścicielkami kolejek klastrowych o tej samej nazwie (gdzie działa instancja programu przykładowego monitorowania kolejek klastra), oraz dowolny menedżer kolejek, w którym aplikacja otworzyła kolejkę klastra w celu umieszczenia komunikatów w kolejce w ciągu ostatnich 30 dni.

Zmiany są przetwarzane asynchronicznie w klastrze. Oznacza to, że po każdej zmianie różne menedżery kolejek w klastrze mogą mieć różne widoki konfiguracji na określony czas.

Przykładowy program monitorujący kolejkę klastra jest odpowiedni tylko dla systemów, w których aplikacje używają rzadko podłączanych lub odłączanych aplikacji, na przykład długotrwałe aplikacje korzystające z pamięci. Jeśli używane do monitorowania systemów, w których aplikacje konsumowane są przyłączane tylko przez krótkie okresy, opóźnienie związane z dystrybucją aktualizacji konfiguracji może spowodować, że menedżery kolejek w klastrze będą mieć niepoprawny widok kolejek, do których przyłączone są konsumenci. To opóźnienie może spowodować niepoprawne kierowanie komunikatów.

W przypadku monitorowania wielu kolejek stosunkowo niski wskaźnik zmian w przyłączonych konsumentach we wszystkich kolejkach może zwiększyć ruch konfiguracji klastra w klastrze. Zwiększone natężenie ruchu w konfiguracji klastra może spowodować nadmierne obciążenie jednego lub większej liczby następujących menedżerów kolejek.

- Menedżery kolejek, w których uruchomiony jest przykładowy program monitorowania kolejki klastra
- Menedżery kolejek pełnego repozytorium
- Menedżer kolejek z połączonym aplikacją, która umieszcza komunikaty w kolejce.
- Menedżer kolejek, który jest właścicielem kolejki lokalnej o tej samej nazwie w tym samym klastrze

Należy ocenić użycie procesora w pełnej kolejce menedżerów kolejek repozytorium. Dodatkowe wykorzystanie procesora jest widoczne jako ruch komunikatów w pełnej kolejce repozytorium SYSTEM.CLUSTER.COMMAND.QUEUE. Jeśli komunikaty są zbudowane w tej kolejce, oznacza to, że menedżery kolejek pełnego repozytorium nie mogą nadążać za szybkością zmiany konfiguracji klastra w systemie.

Jeśli wiele kolejek jest monitorowanych przez program przykładowy monitorujący kolejkę klastra, jest to ilość pracy wykonywana przez program przykładowy i menedżer kolejek. Praca ta jest wykonywana, nawet wtedy, gdy nie ma żadnych zmian w przyłączonych konsumentach. Argument **-i** można zmodyfikować, aby zmniejszyć użycie procesora w programie przykładowym w systemie lokalnym, zmniejszając częstotliwość cyklu monitorowania.

Aby pomóc w wykrywaniu nadmiernego działania, przykładowy program monitorujący kolejkę klastra zgłasza średni czas przetwarzania dla każdego okresu odpytywania, czas przetwarzania i liczbę zmian w konfiguracji. Raporty są dostarczane w komunikacie informacyjnym, **CLM0045I**, co 30 minut lub co 600 interwałów odpytywania, w zależności od tego, co nastąpi wcześniej.

Wymagania dotyczące użycia monitorowania kolejki klastra

Przykładowy program monitorujący kolejkę klastra ma wymagania i ograniczenia. Można zmodyfikować przykładowy kod źródłowy, aby zmienić niektóre z tych ograniczeń, w jaki sposób może być używany. Przykłady wymienione w tej sekcji szczegółów modyfikacji, które mogą zostać wprowadzone.

- Przykładowy program monitorujący kolejkę klastra jest przeznaczony do monitorowania kolejek, w których aplikacje konsumujące są przyłączone lub nie są przyłączone. Jeśli w systemie są używane aplikacje, które często są przyłączane 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 jest zależny od bazowego systemu IBM MQ i technologii klastrowych. Liczba monitorowanych kolejek, częstotliwość monitorowania oraz częstotliwość zmian stanu poszczególnych kolejek wpływa na obciążenie systemu. Czynniki te należy wziąć pod uwagę przy wybieraniu kolejek, które mają być monitorowane, oraz przedział czasu odpytywania w monitorowaniu.
- Instancja przykładowego programu 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 przykładowego programu z menedżerami kolejek w klastrze, które nie są właścicielkami kolejek.
- Przykładowy program monitorujący kolejkę klastra musi być uruchomiony z odpowiednimi uprawnieniami dostępu do wszystkich wymaganych zasobów produktu IBM MQ . Na przykład składnia
  - Menedżer kolejek, z którym ma zostać nawiązane połączenie
  - SYSTEM.ADMIN.COMMAND.QUEUE
  - Wszystkie kolejki, które mają być monitorowane, gdy wykonywane jest przesyłanie komunikatów
- Serwer komend musi być uruchomiony dla każdego menedżera kolejek z podłączonym programem przykładowym monitorowania kolejki klastra.
- Każda instancja przykładowego programu monitorowania kolejki klastra wymaga wyłącznego użycia lokalnej (nieklastrowej) kolejki w menedżerze kolejek, z którą jest on połączony. Ta kolejka lokalna jest używana do sterowania programem przykładowym i do odbierania komunikatów odpowiedzi z zapytań wykonanych na serwerze komend menedżera kolejek.
- Wszystkie kolejki, które mają być monitorowane przez pojedynczą instancję przykładowego programu monitorowania kolejki klastra, muszą znajdować się w tym samym klastrze. Jeśli menedżer kolejek zawiera kolejki w wielu klastrach, które wymagają monitorowania, wymagane jest wiele instancji programu przykładowego. Każda instancja wymaga lokalnej kolejki na potrzeby komunikatów sterujących i odpowiedzi.
- Wszystkie kolejki, które mają być monitorowane, muszą znajdować się w jednym klastrze. Kolejki skonfigurowane do korzystania z listy nazw klastrów nie są monitorowane.
- Włączenie przesyłania komunikatów z nieaktywnych kolejek jest opcjonalne. Odnosi się do wszystkich kolejek monitorowanych przez instancję przykładowego programu monitorowania kolejki klastra. Jeśli tylko podzbiór monitorowanych kolejek wymaga włączenia przesyłania komunikatów, konieczne jest zastosowanie dwóch instancji przykładowego programu monitorowania kolejki klastra. Jeden program przykładowy ma włączone przesyłanie komunikatów, a drugi ma wyłączony transfer komunikatów. Każda instancja programu przykładowego wymaga kolejki lokalnej na potrzeby komunikatów sterujących i odpowiedzi.
- Równoważenie obciążenia klastra IBM MQ będzie domyślnie wysyłać komunikaty do instancji kolejek klastrowych znajdujących się w tym samym menedżerze kolejek, z którym połączona jest aplikacja umieszczana w klastrze. Ta opcja musi być wyłączona, gdy kolejka lokalna jest nieaktywna w następujących okolicznościach:
  - Wprowadzanie aplikacji łączy się z menedżerami kolejek, które są właścicielkami nieaktywnej kolejki, która jest monitorowana
  - Komunikaty w kolejce są przesyłane z nieaktywnych kolejek do aktywnych kolejek.

  Lokalne preferencje równoważenia obciążenia w kolejce można wyłączyć statycznie, ustawiając wartość parametru CLWLUSEQ na ANY. W tej konfiguracji komunikaty umieszczane w kolejkach lokalnych są dystrybuowane do lokalnych i zdalnych instancji kolejek w celu zrównoważenia obciążenia, nawet jeśli istnieją lokalne aplikacje korzystające z pamięci masowej. Alternatywnie, przykładowy program monitorujący kolejkę klastra można skonfigurować w taki sposób, aby tymczasowo ustawiał wartość **CLWLUSEQ** na wartość ANY , podczas gdy kolejka nie ma przyłączonych konsumentów, co powoduje, że tylko komunikaty lokalne będą wysyłane do lokalnych instancji kolejki, gdy ta kolejka jest aktywna.
- System IBM MQ i aplikacje nie mogą używać **CLWLPRTY** dla kolejek, które mają być monitorowane, ani kanałów, które są używane. W przeciwnym razie działania przykładowego programu monitorowania kolejki klastra w atrybutach kolejki produktu **CLWLPRTY** mogą mieć niepożądane efekty.

- Przykładowy program monitorujący kolejkę klastra rejestruje informacje o środowisku wykonawczym w zestawie plików raportów. Katalog do przechowywania tych raportów jest wymagany, a przykładowy program monitorujący kolejkę klastra musi mieć uprawnienia do zapisu w tym programie.

AMQSCLM: Przygotowywanie i uruchamianie przykładu
Przykład monitorowania kolejki klastra można uruchomić lokalnie w połączeniu z menedżerem kolejek lub jako klient połączony za pośrednictwem kanału. Próbka powinna być uruchamiana za każdym razem, gdy menedżer kolejek jest uruchomiony, gdy działa lokalnie, można go skonfigurować jako usługę menedżera kolejek w celu automatycznego uruchamiania i zatrzymywania przykładu z menedżerem kolejek.

Zanim rozpoczniesz

Przed uruchomieniem przykładu monitorowania kolejki klastra należy wykonać następujące kroki.

1. Utwórz kolejkę roboczą dla każdego menedżera kolejek w celu wewnętrznego użycia przykładu.

   Każda instancja przykładu wymaga lokalnej kolejki nieklastrowego 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 parametrów **MAXDEPTH** i **MAXMSGL** można pozostawić jako domyślne.
2. Utwórz katalog dla dzienników komunikatów o błędach i informacji.

   W tym przykładzie komunikaty diagnostyczne są zapisywane 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 przykład monitorowania kolejki klastra jako usługę IBM MQ .

   Aby monitorować kolejki, próbka musi być zawsze uruchomiona. Aby upewnić się, że próbka monitorowania kolejki klastra jest zawsze uruchomiona, można zdefiniować przykład jako usługę menedżera kolejek. Zdefiniowanie przykładu jako usługi oznacza, że menedżer AMQSCLM jest uruchamiany podczas uruchamiania menedżera kolejek. W tym celu można użyć następującego przykładu, aby zdefiniować przykład monitorowania kolejki klastra jako usługę 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 zatrzymana w tym samym czasie co menedżer kolejek.

Definicja	Opis
servtype	Wskazuje, że obiekt usługi serwera, co oznacza tylko jedną instancję, może być wykonywany w danym momencie dla tego menedżera kolejek.
startcmd	Określa lokalizację i nazwę programu.
startarg	Określa argumenty próbki. Należy zwrócić uwagę na użycie parametru *+ QMNAME +*. Nazwa menedżera kolejek jest zastępowana automatycznie.
stdout	Pełna nazwa pliku, do którego przekierowuje się standardowe wyjście danych. Próbka zapisuje do tego pliku tylko komunikaty potwierdzające, że próba została zakończona. Przykład ten jest taki, ponieważ standardowy plik błędów został już zamknięty we wcześniejszej fazie przykładowego procesu zakończenia.
stderr	Pełna nazwa pliku, do którego przekierowane są standardowe wyjście błędów. Próbka zapisuje komunikaty o błędach w standardowym pliku błędów przed zakończeniem próby.

O tym zadaniu

To zadanie umożliwia uruchamianie i zatrzymywanie przykładu monitorowania kolejki klastra na różne sposoby. Umożliwia on również uruchamianie przykładu w trybie generującym pliki raportów zawierające informacje statystyczne na temat monitorowanych kolejek.

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, które mogą być używane wraz z próbką monitorowania kolejki klastra wraz z dodatkowymi informacjami na temat każdego z nich.

Argument	Zmienna	Dalsze informacje
-m	**QMgrName**	Menedżer kolejek do monitorowania.
-c	**ClusterName**	Klaster zawierający kolejki do monitorowania.
-q	**QNameMask**	Kolejka lub kolejki do monitorowania. Końcowy * monitoruje wszystkie kolejki o nazwach, które są zgodne z zero lub więcej znaków końcowych.
-f	**QListFile**	Pełna ścieżka i nazwa pliku zawierającego listę nazw kolejek lub masek nazw kolejek, które mają być monitorowane. Plik musi zawierać jedną nazwę kolejki/maskę na wiersz. Można określić wartość **-q** lub **-f**, ale nie obie jednocześnie.
-r	**MonitorQName**	Kolejka lokalna używana wyłącznie przez przykład.
-l	**ReportDir**	Ścieżka do katalogu, w którym mają być zapisywane rejestrowane komunikaty informacyjne w zestawie zawijania[9]pliki raportów.
-t		(Opcjonalne) Umożliwia przesyłanie komunikatów w kolejce z nieaktywnych kolejek lokalnych do aktywnych kolejek. Jeśli ta opcja nie jest włączona, tylko nowe komunikaty wprowadzane do klastra są dynamicznie kierowane do aktywnych instancji kolejki.
-u	**ActiveVal**	(Opcjonalnie) Automatycznie przełącza właściwość **CLWLUSEQ** monitorowanej instancji kolejki na ANY , gdy jest ona nieaktywna, oraz na wartość **ActiveVal** , gdy jest aktywna. **ActiveVal** może mieć wartość LOCAL lub QMGR. Jeśli ten argument nie zostanie ustawiony w systemie, w którym aplikacje łączą się z tym samym menedżerem kolejek lub jeśli włączono przesyłanie komunikatów, to monitorowane kolejki muszą mieć wartość **CLWLUSEQ** ANYlub QMGR z menedżerem kolejek, który ma wartość ANY.
-i	**Interval**	(Opcjonalnie) Przedział czasu (w sekundach), w którym monitor sprawdza kolejki. Wartość domyślna to 300 sekund (5 minut).
-d		(Opcjonalnie) Włącza dodatkowe wyjście diagnostyczne. Dane wyjściowe debugowania mogą być przydatne podczas początkowego konfigurowania systemu lub podczas pracy z przykładowym kodem.
-s		(Opcjonalnie) Włącza minimalny wynik statystyczny na przedział czasu.
-v		(Opcjonalnie) W uzupełnieniu do plików raportu, informacje o raportach dziennika są dostępne w programie `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 każdego menedżera kolejek i kombinacji kolejki generowany jest plik dziennika o stałej wielkości, który po zapełnieniu jest nadpisywany. Program rejestrujący zawsze zapisuje w tym samym pliku, a także zachowuje dwie poprzednie wersje tego pliku.

```
ABC
ABD
```

Procedura

1. Uruchom próbkę monitorowania kolejki klastra. Przykład można uruchomić w jeden z następujących sposobów:
   - Użyj wiersza komend z odpowiednimi autoryzacjami użytkownika.
   - Użyj komendy MQSC **START SERVICE** , jeśli przykład został skonfigurowany jako usługa IBM MQ .

   Lista argumentów jest taka sama w obu przypadkach.

   Próbka nie uruchamia monitorowania kolejek przez 10 sekund po zainicjowaniu programu. To opóźnienie umożliwia aplikacjom konsumowanie najpierw połączenia się z monitorowanymi kolejkami, zapobiegając niepotrzebnym zmianom stanu aktywnego kolejki.
2. Zatrzymaj przykład monitorowania kolejki klastra. Przykład jest automatycznie zatrzymywany, gdy menedżer kolejek jest zatrzymany, zatrzymywany, wygaszany lub jeśli połączenie z menedżerem kolejek jest zerwane. Istnieją sposoby zatrzymania próby bez zakończenia menedżera kolejek:
   - Skonfiguruj kolejkę lokalną używaną wyłącznie przez przykład, aby wyłączyć funkcję Get.
   - Wyślij komunikat z **CorrelId** o wartości `"STOP CLUSTER MONITOR\0\0\0\0"`do kolejki lokalnej używanej wyłącznie przez przykład.
   - Zakończ proces przykładowy. Może to spowodować utratę nietrwałych komunikatów przesyłanych do aktywnych kolejek. Może to również spowodować, że kolejka lokalna używana przez próbkę jest wstrzymana przez liczbę sekund po zakończeniu. Ta sytuacja uniemożliwia natychmiastowe uruchomienie nowej instancji przykładu monitorowania kolejki klastra.

   Jeśli próbka została uruchomiona jako usługa IBM MQ , produkt **STOP SERVICE** nie ma żadnego efektu. Możliwe jest użycie jednej z metod zakończenia opisanych jako skonfigurowany mechanizm **STOP SERVICE** w menedżerze kolejek.

Co dalej

Sprawdź status przykładu.

Jeśli raportowanie jest włączone, można przejrzeć pliki raportów w celu uzyskania statusu. Użyj następującej komendy, aby przejrzeć najbardziej aktualny plik raportu:

```
QMgrName.ClusterName.RPT01.LOG
```

Aby przejrzeć starsze pliki raportów, użyj następujących komend:

```
QMgrName.ClusterName.RPT02.LOG
QMgrName.ClusterName.RPT03.LOG
```

Wielkość plików raportu może wynosić maksymalnie ok. 1 MB. Gdy plik RPT01 zostanie zapełniony, tworzony jest nowy plik RPT01 . Nazwa starego pliku RPT01 zostanie zmieniona na RPT02. Nazwa RPT02 została zmieniona na RPT03. Stary RPT03 jest odrzucany.

Przykład tworzy komunikaty informacyjne w następujących sytuacjach:

- przy starcie
- na zakończenie
- gdy oznacza kolejkę **ACTIVE** lub **INACTIVE**
- w przypadku ponownego przesłania komunikatów z nieaktywnej kolejki do aktywnej instancji lub instancji

W przykładzie zostanie utworzony komunikat o błędzie *CLMnnnnE* w celu zgłoszenia problemu, który wymaga uwagi.

Co 30 minut próbka raportuje średni czas przetwarzania dla przedziału czasu odpytywania i czas przetwarzania. Te informacje są przechowywane w komunikacie CLM0045I.

Gdy komunikaty statystyczne są włączone **-s**, przykładowy raport przedstawia następujące informacje statystyczne o każdej kontroli kolejki:

- Czas przetwarzania kolejek (w milisekundach)
- Liczba sprawdzanych kolejek
- Liczba wprowadzonych zmian aktywnych/nieaktywnych
- Liczba przesłanych wiadomości

Te informacje są zgłaszane w komunikacie CLM0048I.

Pliki raportów mogą gwałtownie rosnąć w trybie debugowania i szybko zawijać. W tej sytuacji limit wielkości 1 MB dla poszczególnych plików może zostać przekroczony.

AMQSCLM: Rozwiązywanie problemów
W poniższych sekcjach znajdują się informacje o scenariuszach, które mogą wystąpić podczas korzystania z przykładu. Dostępne są informacje na temat potencjalnych wyjaśnień dotyczących scenariusza oraz opcje dotyczące sposobu jego rozwiązania.

Scenariusz: AMQSCLM nie jest uruchamiany

Potencjalne wyjaśnienie: Niepoprawna składnia.

Działanie: Sprawdź standardowe wyjście błędów dla poprawnej składni.

Potencjalne wyjaśnienie: Menedżer kolejek nie jest dostępny.

Działanie: Sprawdź plik raportu dla identyfikatora komunikatu CLM0010E.

Potencjalne wyjaśnienie: Nie można otworzyć ani utworzyć pliku lub plików raportu.

Działanie: Podczas inicjowania sprawdź standardowe wyjście błędów dla komunikatów o błędach.

Scenariusz: AMQSCLM nie zmienia kolejki na ACTIVE lub INACTIVE

Potencjalne wyjaśnienie: Kolejka nie znajduje się na liście kolejek do monitorowania

Działanie: Sprawdź wartości parametrów **-q** i **-f** .

Potencjalne wyjaśnienie: Kolejka nie jest kolejką lokalną w poprawnym klastrze.

Działanie: Sprawdź, czy kolejka jest lokalna i czy w poprawnym klastrze.

Potencjalne wyjaśnienie: Menedżer AMQSCLM nie działa dla tego menedżera kolejek i klastra.

Działanie: Uruchom komendę AMQSCLM dla odpowiedniego menedżera kolejek i klastra.

Potencjalne wyjaśnienie: Kolejka jest pozostawiona NIEAKTYWNE, **CLWLPRTY** = 0, ponieważ nie ma ona konsumentów. Alternatywnie, pozostanie AKTYWNE **CLWLPRTY** > =1, ponieważ ma co najmniej 1 konsumenta.

Działanie: Sprawdź, czy aplikacje korzystające z aplikacji są przyłączone do kolejki.

Potencjalne wyjaśnienie: Serwer komend menedżera kolejek nie jest uruchomiony.

Działanie: Sprawdź, czy w plikach raportów nie występują błędy.

Scenariusz: komunikaty nie są kierowane przez kolejki INACTIVE

Potencjalne wyjaśnienie: Komunikaty są umieszczane bezpośrednio w menedżerze kolejek, który jest właścicielem nieaktywnej kolejki, a wartością parametru **CLWLUSEQ** kolejki jest inny niż ANY, a argument **-u** nie jest używany dla komendy AMQSCLM.

Działanie: Sprawdź wartość **CLWLUSEQ** odpowiedniego menedżera kolejek lub upewnij się, że argument **-u** jest używany dla AMQSCLM.

Potencjalne wyjaśnienie: W żadnym menedżerze kolejek nie ma aktywnych kolejek. Komunikaty są równomiernie zbalansowane we wszystkich nieaktywnych kolejkach aż do momentu, gdy kolejka stanie się aktywna.

Działanie: Sprawdź status kolejek we wszystkich menedżerach kolejek.

Potencjalne wyjaśnienie: Komunikaty są umieszczane w innym menedżerze kolejek w klastrze do tego, który jest właścicielem nieaktywnej kolejki, a zaktualizowana wartość **CLWLPRTY** równa 0 nie jest propagowana do menedżera kolejek umieszczanego aplikacji.

Działanie: Sprawdź, czy kanały klastra między monitorowanym menedżerem kolejek i pełnym menedżerem kolejek repozytorium są uruchomione. Sprawdź, czy kanały między umieszczonym menedżerem kolejek a menedżerem kolejek pełnego repozytorium są uruchomione. Sprawdź dzienniki błędów monitorowanych, umieszczających i pełnych menedżerów kolejek repozytorium.

Potencjalne wyjaśnienie: Instancje kolejek zdalnych są aktywne (CLWLPRTY=1), ale komunikaty nie mogą być kierowane do tych instancji kolejek, ponieważ kanał nadawczy klastra z lokalnego menedżera kolejek nie jest uruchomiony.

Działanie: Sprawdź status kanałów nadawczych klastra z lokalnego menedżera kolejek do zdalnego menedżera kolejek lub menedżerów z aktywną instancją kolejki.

Scenariusz: AMQSCLM nie przesyła komunikatów z nieaktywnej kolejki

Potencjalne wyjaśnienie: przesyłanie komunikatów nie jest włączone (**-t**).

Działanie: Upewnij się, że przesyłanie komunikatów jest włączone (**-t**).

Potencjalne wyjaśnienie: Kolejka nie znajduje się na liście kolejek, które mają być monitorowane.

Działanie: Sprawdź wartości parametrów **-q** i **-f** .

Potencjalne wyjaśnienie: W przypadku tego lub innych menedżerów kolejek w klastrze, które są właścicielkami tej samej kolejki, nie jest uruchomiony menedżer AMQSCLM.

Działanie: Uruchom komendę AMQSCLM.

Potencjalne wyjaśnienie: W kolejce jest **CLWLUSEQ** = LOCAL lub **CLWLUSEQ** = QMGR, a argument **-u** nie jest ustawiony.

Działanie: Ustaw parametr **-u** lub zmień konfigurację kolejki lub menedżera kolejek na wartość ANY.

Potencjalne wyjaśnienie: W klastrze nie ma aktywnych instancji kolejki.

Działanie: Sprawdź, czy w przypadku instancji kolejki wartość **CLWLPRTY** jest równa 1 lub większa.

Potencjalne wyjaśnienie: Instancje kolejek zdalnych mają konsumenty (**IPPROCS** > = 1), ale są nieaktywne w tych menedżerach kolejek (**CLWLPRTY** = 0), ponieważ AMQSCLM nie monitoruje tych zdalnych instancji.

Działanie: Upewnij się, że menedżer AMQSCLM jest uruchomiony w tych menedżerach kolejek i/lub 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 nieaktywne w lokalnym menedżerze kolejek (**CLWLPRTY** = 0). Ta sytuacja wynika ze zaktualizowanej wartości **CLWLPRTY** , która nie jest propagowana do tego menedżera kolejek.

Działanie: Upewnij się, że zdalne menedżery kolejek są połączone z co najmniej jednym z menedżerów kolejek pełnego repozytorium w klastrze. Upewnij się, że menedżery kolejek pełnego repozytorium działają poprawnie. Sprawdź, czy kanały między pełnymi menedżerami kolejek repozytorium i monitorowanymi menedżerami kolejek są uruchomione.

Potencjalne wyjaśnienie: Komunikaty nie są zatwierdzane, dlatego nie są dostępne do pobrania.

Działanie: Sprawdź, czy aplikacja wysyłający działa poprawnie.

Potencjalne wyjaśnienie: AMQSCLM nie ma dostępu do kolejki lokalnej, w której komunikaty są umieszczane w kolejce.

Działanie: Sprawdź, czy program AMQSCLM jest uruchomiony jako użytkownik z wystarczającą autoryzacją do uzyskania dostępu do kolejki.

Potencjalne wyjaśnienie: Serwer komend menedżera kolejek nie jest uruchomiony.

Działanie: Uruchom serwer komend menedżera kolejek.

Potencjalne wyjaśnienie: Wystąpił błąd AMQSCLM.

Działanie: Sprawdź, czy w plikach raportów nie występują błędy.

Potencjalne wyjaśnienie: Instancje kolejki zdalnej są aktywne (CLWLPRTY=1), ale komunikaty nie mogą być przesyłane do tych instancji kolejek, ponieważ kanał nadawczy klastra z lokalnego menedżera kolejek nie jest uruchomiony. Często towarzyszy temu ostrzeżenie CLM0030W w dzienniku raportów amqsclm.

Działanie: Sprawdź status kanałów nadawczych klastra z lokalnego menedżera kolejek do zdalnego menedżera kolejek lub menedżerów z aktywną instancją kolejki.

ULW *Przykładowy program do wyszukiwania punktów końcowych połączenia (CEPL)*

IBM MQ Przykład wyszukiwania punktów końcowych połączenia udostępnia prosty, a jednocześnie wydajny moduł obsługi wyjścia, który oferuje użytkownikom produktu IBM MQ sposób pobierania definicji połączeń z repozytorium LDAP, takiego jak Tivoli Directory Server.

Aby można było używać programu CEPL, należy zainstalować klienta Tivoli Directory Server v6.3 .

Do korzystania z tego przykładu wymagana jest praktowa wiedza na temat administrowania produktem IBM MQ na obsługiwanych platformach.

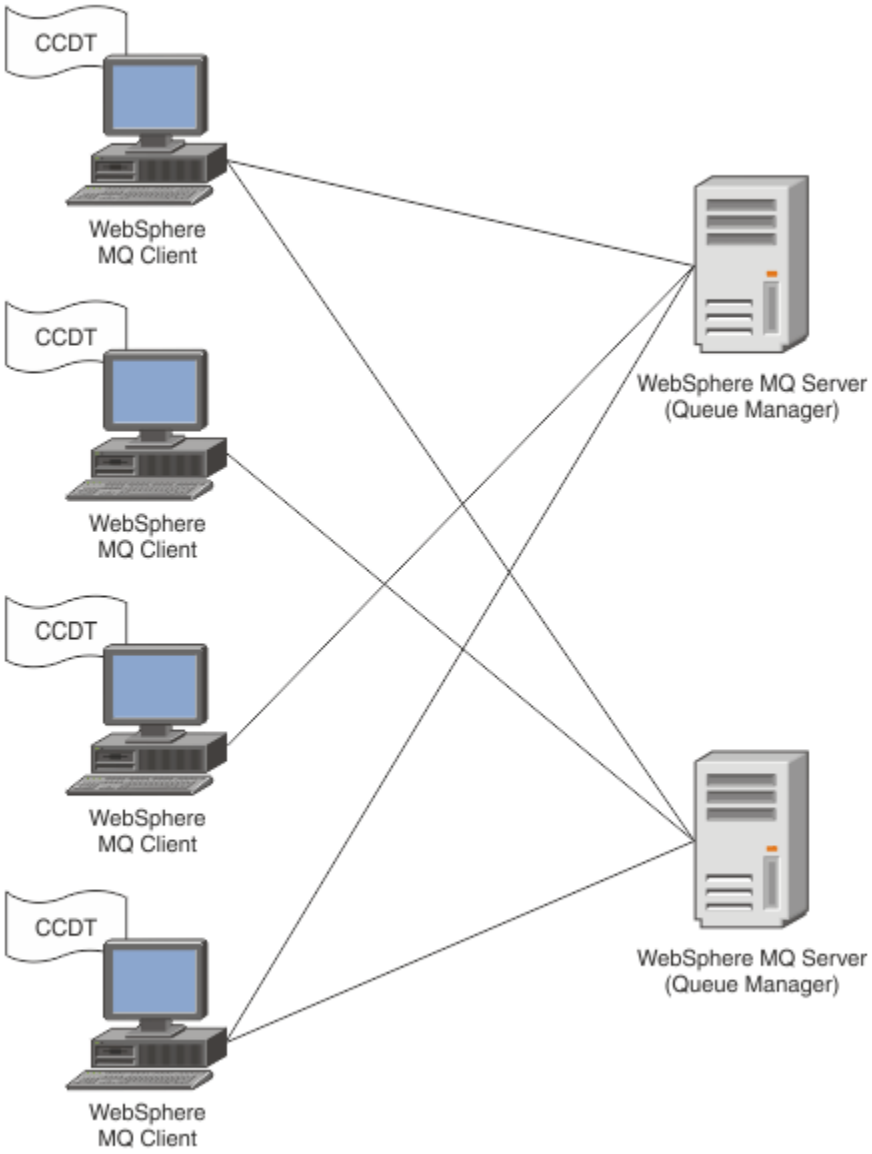
Windows Solaris Linux AIX *Wprowadzenie*

Skonfiguruj globalne repozytorium, na przykład katalog LDAP (Lightweight Directory Access Protocol), w celu zapisania definicji połączenia klienta w celu konserwacji i administrowania pomocą.

Korzystanie z aplikacji klienta IBM MQ w celu nawiązania połączenia z menedżerem kolejek za pośrednictwem tabeli definicji połączeń klienta (Client Connection Definition Table-CCDT).

Pakiet CCDT jest tworzony za pośrednictwem standardowego interfejsu administracyjnego MQSC produktu IBM MQ . Aby można było utworzyć definicje połączeń klientów, użytkownik musi być połączony z menedżerem kolejek, nawet jeśli dane zawarte w definicji nie są ograniczone do menedżera kolejek.

Wygenerowany plik CCDT musi być rozłożony ręcznie między maszynami klienckim i aplikacjami.

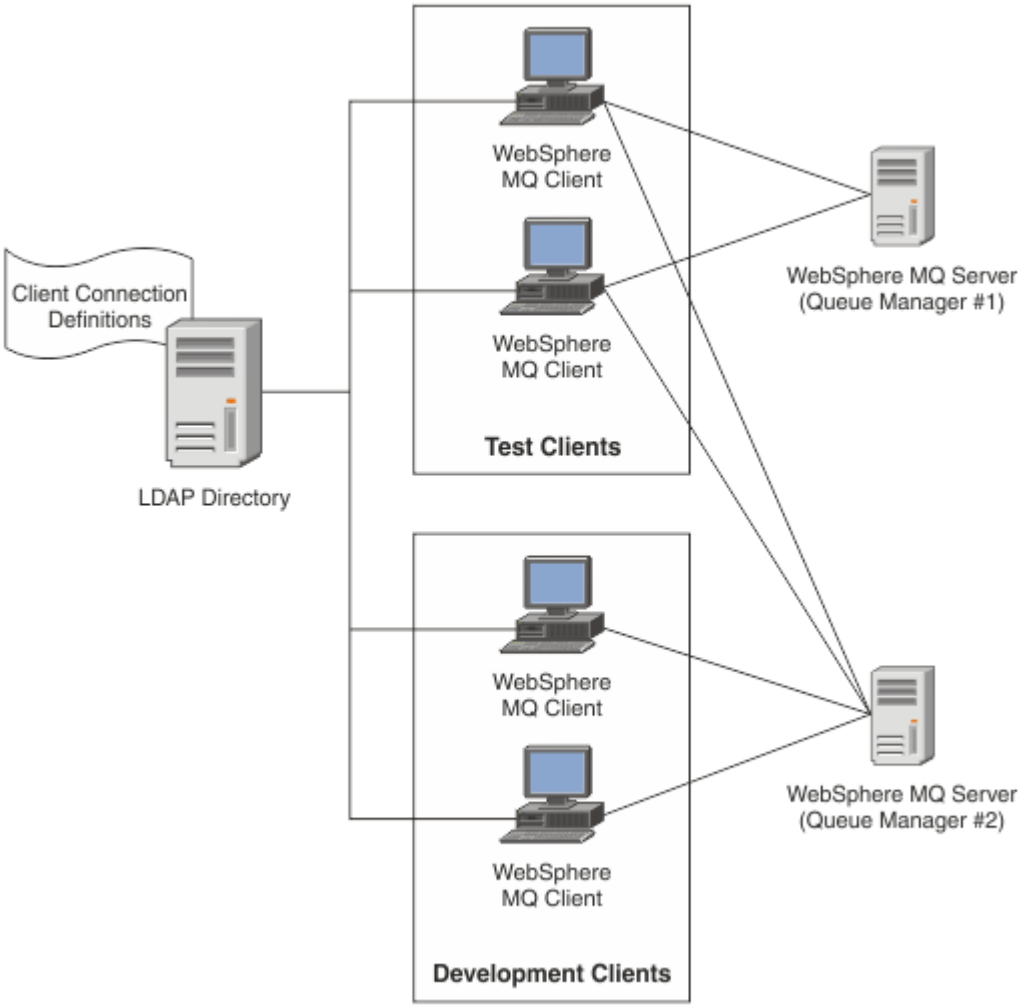


Plik CCDT musi być dystrybuowany do każdego klienta IBM MQ . W przypadku, gdy tysiące klientów może istnieć lokalnie lub globalnie, wkrótce stanie się to trudne do utrzymania i administrowania. Potrzebne jest bardziej elastyczne podejście, aby zapewnić, że każdy klient będzie miał do dyspozycji poprawne definicje klientów.

Jednym z takich podejść jest przechowywanie definicji połączeń klienta w repozytorium globalnym, takim jak katalog LDAP (Lightweight Directory Access Protocol). Katalog LDAP może również udostępniać dodatkowe zabezpieczenia, indeksowanie i wyszukiwanie, umożliwiając każdemu klientowi dostęp tylko do tych definicji połączeń, które są z nimi związane.

Katalog LDAP można skonfigurować w taki sposób, aby dla określonych grup użytkowników były dostępne tylko konkretne definicje. Na przykład klienty testowe mogą uzyskiwać dostęp zarówno do menedżera

kolejek #1 , jak i do produktu #2, podczas gdy klienty programistyczne mają dostęp tylko do menedżera



kolejek #2 .

Moduł wyjścia może wyszukać repozytorium LDAP, na przykład IBM Tivoli Directory Server, w celu pobrania definicji kanałów. Korzystając z tych definicji połączeń, aplikacja kliencka IBM MQ może nawiązać połączenie z menedżerem kolejek.

Moduł wyjścia jest modułem wyjścia typu pre-connect, który umożliwia uzyskanie definicji kanału podczas wywołania MQCONN/MQCONNX z repozytorium LDAP.

Moduł wyjścia i schemat mogą być implementowane przez:

- Klienci, którzy już zbudowali bazę umiejętności, korzystając z istniejącej technologii opartej na pliku CCDT i chcą zmniejszyć koszty administrowania i dystrybucji.
- Dotychczasowi klienci, którzy już zatrudniają własną technologię dystrybuowania definicji połączeń klientów.
- Nowi lub obecni klienci, którzy obecnie nie korzystają z żadnego typu rozwiązania do połączenia z klientem i chcą korzystać z opcji oferowanych przez produkt IBM MQ.
- Nowi lub obecni klienci, którzy chcą bezpośrednio wykorzystać lub dostroić swój model przesyłania komunikatów w sposób wstawiany do dowolnej bieżącej architektury biznesowej LDAP.

ULW *Obsługiwane środowiska*

Przed uruchomieniem przykładu wyszukiwania punktu końcowego połączenia sprawdź, czy istnieje obsługiwany system operacyjny i odpowiednie oprogramowanie.

Przykładowy program do wyszukiwania punktów końcowych połączenia IBM MQ wymaga następującego oprogramowania:

- IBM WebSphere MQ 7.0lub nowsza
- Tivoli Directory Server V6.3 Client lub nowsza wersja

Obsługiwane systemy operacyjne:

1. Windows (7/8/2008/2012)
2. Solaris (SPARC i x86-64)
3. AIX
4. Linux
   - RHEL v4 i v5 w systemie System p
   - SUSE v9 i v10 w systemie System p
   - RHEL v4 i v5 x86-64 w wersji 32-bitowej i 64-bitowej
   - SUSE v9 i v10 x86-64 w wersji 32-bitowej i 64-bitowej

Uwaga: Przykład nie jest dostępny dla następujących platform:

- z/OS
- IBM i

ULW *Instalowanie i konfigurowanie*
Instalowanie i konfigurowanie modułu obsługi wyjścia i schematu punktu końcowego połączenia.

Instalowanie modułu wyjścia

Podczas instalacji produktu IBM MQmoduł obsługi wyjścia jest instalowany w ramach produktu `tools/samples/c/preconnexit/bin`. W przypadku platform 32-bitowych moduł wyjścia musi być skopiowany do produktu `exit/`*`installation_name`*`/` , zanim będzie można go użyć. W przypadku platform 64-bitowych moduł wyjścia musi zostać skopiowany do katalogu exit64/*nazwa_instalacji*/zanim będzie mógł zostać użyty.

Instalowanie schematu punktu końcowego połączenia

Wyjście korzysta ze schematu punktu końcowego połączenia `ibm-amq.schema`. Plik schematu musi zostać zaimportowany do dowolnego serwera LDAP, aby można było użyć wyjścia. Po zaimportowaniu schematu należy dodać wartości dla atrybutów.

Poniżej przedstawiono przykład importowania schematu punktu końcowego połączenia. W przykładzie założono, że program IBM Tivoli Directory Server (ITDS) jest używany.

- Upewnij się, że serwer IBM Tivoli Directory Server jest uruchomiony, a następnie skopiuj plik `ibm-amq.schema` do serwera ITDS lub skopiuj go na serwer ITDS.
- Na serwerze ITDS wprowadź następującą komendę, aby zainstalować schemat w składnicy ITDS, gdzie *ID LDAP* i *Hasło LDAP* to główna nazwa wyróżniająca i hasło 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
```

Szczegółowe informacje na temat importowania pliku schematu można znaleźć w dokumentacji serwera LDAP.

Konfiguracja

Do pliku konfiguracyjnego klienta musi zostać dodana nowa sekcja o nazwie PreConnect , na przykład `mqclient.ini`. Sekcja PreConnect zawiera następujące słowa kluczowe:

Moduł: Nazwa modułu zawierającego kod wyjścia API. Jeśli w tym polu znajduje się pełna ścieżka do modułu, jest on używany w takiej postaci. W przeciwnym razie przeszukiwany jest folder `exit` lub `exit64` w instalacji produktu IBM MQ .

Funkcja: nazwa punktu wejścia funkcjonalnego w bibliotece, która zawiera kod wyjścia PreConnect . Definicja funkcji jest zgodna z prototypem MQ_PRECONNECT_EXIT.

Dane: identyfikator URI repozytorium LDAP zawierającego definicje kanałów.

Poniższy fragment kodu stanowi przykład zmian wymaganych w pliku `mqclient.ini` .

```
PreConnect:
Module=amqlcelp
Function=PreConnectExit
Data=ldap://myLDAPServer.com:389/cn=wmq,ou=ibm,ou=com
Sequence=1
```

ULW *Przegląd wyjścia i schematu*

Składnia i parametry używane do nawiązania połączenia z menedżerem kolejek.

IBM MQ 9.1 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, który zawiera implementację składni funkcji. Następnie wywołuje funkcję wyjścia w celu pobrania definicji kanałów. Pobrane definicje kanałów są następnie używane do nawiązania połączenia z menedżerem kolejek.

Parametry

Parametry pExit

Typ: PMQNXP input/output

Struktura parametru wyjścia PreConnection (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 */
};
```

Nazwa pQMgr

Typ: input/output PMQCHAR

Nazwa menedżera kolejek. Na wejściu parametr ten jest łańcuchem filtru dostarczonym do wywołania MQCONN API za pomocą parametru **QMgrName** . To pole może być puste, jawne lub zawierać

określone znaki wieloznaczne. Pole zostanie zmienione przez wyjście. Parametr ma wartość NULL , gdy wyjście jest wywoływane z MQXR_TERM.

ppConnectOpts

Wpisz: ppConnectOpts input/output

Opcje, które sterują działaniem MQCONNX. Jest to wskaźnik do struktury opcji połączenia MQCNO, która steruje działaniem wywołania interfejsu API MQCONN. Parametr ma wartość NULL , gdy wyjście jest wywoływane z MQXR_TERM. Klient MQI zawsze udostępnia strukturę MQCNO do wyjścia, nawet jeśli nie została ona pierwotnie udostępniona przez aplikację. Jeśli aplikacja udostępnia strukturę MQCNO, klient tworzy duplikat w celu przekazania go do wyjścia, w którym jest on modyfikowany. Klient zachowuje prawo własności do obiektu MQCNO. Tabela MQCD, do której odwołuje się parametr MQCNO, ma pierwszeństwo przed każdą definicją połączenia udostępnionej przez tablicę. Klient używa struktury MQCNO do łączenia się z menedżerem kolejek, a pozostałe są ignorowane.

Kod pComp

Typ: PMQLONG input/output

Kod zakończenia. Wskaźnik do obiektu MQLONG, który odbiera kod zakończenia wyjścia. Musi to być jedna z następujących wartości:

- MQCC_OK -pomyślne zakończenie
- MQCC_WARNING -ostrzeżenie (częściowe zakończenie)
- MQCC_FAILED -wywołanie nie powiodło się

pReason

Typ: PMQLONG input/output

Przyczyna kwalifikowania kodu pComp. Wskaźnik do obiektu MQLONG, który odbiera kod przyczyny wyjścia. Jeśli kodem zakończenia jest MQCC_OK, jedyną poprawną wartością jest: MQRC_NONE -(0, x '000'). Nie ma powodu, aby zgłosić ten kod.

Jeśli kod zakończenia to MQCC_FAILED lub MQCC_WARNING, to funkcja wyjścia może ustawić pole kodu przyczyny na dowolną poprawną wartość MQRC_ *.

ULW *Informacje o kontekście LDAP MQ*

Wyjście korzysta z następującej struktury danych dla informacji kontekstowych.

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 Solaris Linux AIX *Przykładowy kod do budowania wyjścia wyszukiwania punktu końcowego połączenia*

Możliwe jest użycie przykładowych fragmentów kodu do kompilowania źródła w systemach AIX, Linux, Solarisi Windows.

Kompilowanie źródła

Istnieje możliwość skompilowania źródła z dowolnymi bibliotekami klienta LDAP, na przykład IBM Tivoli Directory Server V6.3 . W tej dokumentacji przyjęto założenie, że używane są biblioteki klienta Tivoli Directory Server V6.3 .

Uwaga: Biblioteka wyjścia wstępnego połączenia jest obsługiwana przez następujące serwery LDAP:

- IBM Tivoli Directory Server V6.3
- Novell eDirectory V8.2

Następujące fragmenty kodu opisują sposób kompilowania wyjść:

Windows **Kompilowanie wyjścia na platformie Windows**

W celu kompilowania 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 , podczas kompilowania bibliotek klienta produktu IBM Tivoli Directory Server V6.3 z produktem Microsoft Visual Studio 2012lub późniejszym kompilatorem może zostać wyświetlone ostrzeżenia.

Solaris Linux AIX **Kompilowanie wyjścia w systemach AIX, Linuxlub Solaris**

Następujący fragment kodu jest przeznaczony do kompilowania źródła wyjścia w systemie Linux. Niektóre opcje kompilatora mogą być różne w przypadku produktu AIX lub Solaris.

```
#Make file to build exit
CC=gcc

MQML=/opt/mqm/lib
MQMI=/opt/mqm/inc
TDSI=/opt/ibm/ldap/V6.3/include
XFLAG=-m32

TDSL=/opt/ibm/ldap/V6.3/lib
```

IBM Tivoli Directory Server jest dostarczany zarówno ze statycznymi, jak i dynamicznymi bibliotekami połączeń, ale można używać tylko jednego typu biblioteki. W tym skrypcie przyjęto założenie, 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)
```

ULW *Wywołanie modułu wyjścia PreConnect*

Moduł wyjścia PreConnect może zostać wywołany z trzema różnymi kodami przyczyny: kod przyczyny MQXR_INIT do inicjowania i nawiązywania połączenia z serwerem LDAP, kod przyczyny MQXR_PRECONNECT do pobierania definicji kanałów z serwera LDAP lub kod przyczyny MQXR_TERM, gdy wyjście ma być czyszczone.

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 Dane z sekcji PreConnect w pliku `mqclient.ini` (to znaczy LDAP).

Adres URL LDAP składa się co najmniej z protokołu, nazwy hosta, numeru portu i podstawowej nazwy wyróżniającej dla wyszukiwania. Program obsługi wyjścia analizuje adres URL LDAP zawarty w polu pExitDataPtr , przydziela strukturę kontekstu wyszukiwania LDAP MQNLDAPCTX i zapełnia je odpowiednio. Adres tej struktury jest zapisany w polu pExitUserAreaPtr. Niepowodzenie poprawnego przeanalizowania adresu URL LDAP w wyniku błędu MQCC_FAILED.

W tym momencie wyjście łączy się i łączy z serwerem LDAP za pomocą parametrów **MQNLDAPCTX** . Wynikowe uchwyty interfejsu API LDAP są również przechowywane w tej strukturze.

MQXR_PRECONNECT

Moduł obsługi wyjścia jest wywoływany z kodem przyczyny MQXR_PRECONNECT w celu pobrania definicji kanałów z serwera LDAP.

Program zewnętrzny przeszukuje serwer LDAP w celu uzyskania definicji kanałów zgodnych z podanym filtrem. Jeśli **QMgrNameparameter** zawiera konkretną nazwę menedżera kolejek, to 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), a następnie wyszukiwanie zwraca wszystkie definicje kanałów, dla których atrybut punktu końcowego **ibm-amqIsClientDefault Connection** jest ustawiony na wartość TRUE.

Po pomyślnym wyszukaniu program obsługi wyjścia przygotowuje jedną lub tablicę definicji MQCD i powraca do programu wywołującego.

MQXR_TERM

Wyjście jest wywoływane z tym kodem przyczyny, gdy wyjdź ma zostać wyczyszczona. Podczas tego czyszczenia wyjście rozłącza się z serwerem LDAP i zwalnia całą pamięć przydzieloną i utrzymywana przez wyjście, w tym strukturę MQNLDAPCTX, tablicę wskaźników i wszystkie odwołania MQCD, które są przywoływana przez program zewnętrzny. Wszystkie pozostałe pola są ustawiane na wartości domyślne. Parametry wyjścia **pQMgrName** i **ppConnectOpts** nie są używane podczas wyjścia z kodem przyczyny MQXR_TERM i mogą mieć wartość NULL.

Pojęcia pokrewne

Sekcja PreConnect w pliku konfiguracyjnym klienta

ULW *Schematy LDAP*

Dane połączenia klienta są przechowywane w repozytorium globalnym zwanym katalogiem LDAP (Lightweight Directory Access Protocol). Klient IBM MQ korzysta z katalogu LDAP w celu uzyskania definicji połączeń. Struktura definicji połączeń klienta IBM MQ w katalogu LDAP jest znana jako schemat LDAP. Schemat LDAP to kolekcja definicji typów atrybutów, definicji klas obiektów i innych informacji używanych przez serwer w celu określenia, czy asercja filtru lub wartości atrybutu jest zgodna z atrybutami pozycji, a także czy ma być dozwolone, dodawać i modyfikować operacje.

Zapisywanie danych w katalogu LDAP

Definicje połączeń klienta znajdują się w konkretnej gałęzi w drzewie katalogów, o którym wiadomo, że jest to punkt połączenia. Podobnie jak w przypadku wszystkich innych węzłów w katalogu LDAP, punkt połączenia ma powiązaną nazwę wyróżniającą (DN). Węzła tego można używać jako punktu początkowego dla zapytań, które są wykonywane w katalogu. Użyj filtrowania podczas wysyłania zapytania do katalogu LDAP w celu zwrócenia podzbioru definicji połączeń klienta. Dostęp do poddrzew można ograniczyć w oparciu o uprawnienia nadane w innych częściach drzewa katalogów-na przykład dla 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 numer identyfikatora obiektu (OID), który jednoznacznie identyfikuje obiekt lub atrybut. Wszystkie klasy w schemacie LDAP dziedziczą bezpośrednio lub pośrednio z najwyższego obiektu. Obiekt definicji kanału klienta zawiera atrybuty obiektu najwyższego poziomu. Wszystkie definicje danych LDAP wymagają obiektów i atrybutów:

- Definicje obiektów to kolekcje atrybutów LDAP.
- Atrybuty to typy danych LDAP.

Opis każdego atrybutu i sposób ich odwzorowania na normalne właściwości produktu IBM MQ są opisane w sekcji Atrybuty LDAP.

ULW *LDAP - atrybuty*

Zdefiniowane atrybuty LDAP są specyficzne dla produktu IBM MQ i są odwzorowywać bezpośrednio na właściwości połączenia klienta.

IBM MQ Łańcuch katalogu kanału klienta-atrybuty

Atrybuty łańcucha znaków z ich odwzorowaniem na właściwości produktu IBM MQ są wymienione w poniższej tabeli. Atrybuty mogą zawierać wartości directoryString (UTF-8 zakodowany Unicode, czyli system kodowania bajtów, który zawiera kod IA5/ASCII jako podzbiór). Składnia jest określana na podstawie numeru identyfikacyjnego obiektu (OID).

Tabela 173. Atrybuty łańcucha kanału klienta produktu IBM MQ

Atrybut LDAP	Opis	IBM MQ Właściwość
CN	Nazwa zwykła składająca się z nazwy kanału i nazwy definiującego menedżera kolejek.	
Nazwa IBM-amqChannel	Nazwa definicji kanału.	CHANNEL
Nazwa IBM-amqConnection	Identyfikator połączenia komunikacyjnego.	CONNAME
ibm-amqDescription	Opis kanału.	DESCR
Adres IBM-amqLocal	Lokalny adres komunikacyjny kanału.	LOCLADDR
ibm-amqModeNazwa	Nazwa trybu LU 6.2.	MODENAME
ibm-amqPassword	Hasło, które może być używane.	PASSWORD
ibm-amqQueueManagerName	Nazwa menedżera kolejek lub grupy menedżerów kolejek, do której może zażądać połączenia aplikacja kliencka IBM MQ .	QMNAME
ibm-amqSecurityExitUserData	Dane użytkownika, które są przekazywane do wyjścia zabezpieczeń.	SCYDATA
ibm-amqSecurityExitName	Nazwa programu obsługi wyjścia, który ma być uruchomiony przez wyjście zabezpieczeń kanału.	SCYEXIT

Tabela 173. Atrybuty łańcucha kanału klienta produktu IBM MQ (kontynuacja)		
Atrybut LDAP	**Opis**	**IBM MQ Właściwość**
ibm-amqSslCipherSpec	Pojedyncza specyfikacja CipherSpec dla połączenia TLS.	SSLCIPH
ibm-amqSslPeerName	Sprawdza nazwę wyróżniającą (DN) certyfikatu pochodzącego od menedżera kolejek węzła sieci lub klienta na drugim końcu kanału IBM MQ .	SSLPEER
ibm-amqTransactionProgramName	Nazwa programu transakcyjnego.	TPNAME
Identyfikator ibm-amqUser	Identyfikator użytkownika, który ma być używany przez agenta MCA podczas próby zainicjowania bezpiecznej sesji SNA z użyciem zdalnego agenta MCA.	USERID

Liczba całkowita połączenia klienta IBM MQ -atrybuty

Atrybuty z predefiniowanymi wartościami (na przykład typ wyliczeniowy) są zapisywane jako standardowe liczby całkowite. Wartości te są przechowywane w katalogu LDAP jako wartości całkowite, a nie przez użycie powiązanej nazwy stałej.

Tabela 174. Liczba całkowita katalogu kanału klienta IBM MQ -atrybuty		
Atrybut LDAP	**Opis**	**IBM MQ Właściwość**
ibm-amqConnectionpowinowactwo	Określa, czy aplikacje klienckie, które łączą się wiele razy z tą samą nazwą menedżera kolejek, korzystają z tego samego kanału klienta.	AFFINITY
ibm-amqClientChannelWeight	Waga wpływająca na użycie definicji kanału połączenia klienta.	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-amqSharing-konwersacje	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 boolowką.

Tabela 175. Atrybut boolowski kanału klienta IBM MQ	
Atrybut LDAP	**Opis**
ibm-amqIsClientDefault	Ten atrybut boolowski jest zdefiniowany w celu rozwiązania problemu wyszukiwania pozycji, których atrybut ibm-amqQueueManagerName nie został zdefiniowany.

Atrybuty listy kanałów klienta IBM MQ

Właściwości IBM MQ są przechowywane jako jednowartościowe, rozdzielane przecinkami atrybuty listy w katalogu LDAP. Atrybuty są definiowane w taki sam sposób, jak inne atrybuty łańcuchowe katalogu. Atrybuty listy wraz z ich odwzorowaniem na właściwości produktu IBM MQ są opisane w poniższej tabeli.

Tabela 176. Atrybuty listy kanałów klienta IBM MQ

Atrybut LDAP	Opis	IBM MQ Właściwość
ibm-amqHeaderKompresja	Lista technik kompresji danych nagłówka obsługiwanych przez kanał.	COMPHDR
ibm-amqMessage-kompresja	Lista technik kompresji danych komunikatu obsługiwanych przez kanał.	COMPMSG
ibm-amqSendExitUserData	Dane użytkownika, które są przekazywane do wyjścia wysyłania.	SENDDATA
Nazwa programu ibm-amqSendExitUser	Nazwa programu obsługi wyjścia, który ma być uruchomiony przez wyjście wysyłania kanału.	SENDEXIT
ibm-amqReceiveExitUserData	Dane użytkownika, które są przekazywane do wyjścia odbierania.	RCVDATA
ibm-amqReceiveExitName	Nazwa programu użytkownika obsługi wyjścia, który ma być uruchomiony przez kanał wyjściowy odbierania użytkownika.	RCVEXIT

ULW *Nazwa zwykła*

Nazwa zwykła (CN) składa się z nazwy kanału i nazwy definiującego menedżera kolejek.

Jest to istniejący atrybut.

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, a w wartościach nie jest rozróżniana wielkość liter. Dopasowywanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowywania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania, przy użyciu podłańcucha (na przykład CN=jim *, gdzie CN jest atrybutem) i zawiera jedną lub więcej znaków wieloznacznych.

ULW *Nazwa IBM-amqChannel*

Ten atrybut określa nazwę definicji kanału.

Ten atrybut ma pojedynczą wartość łańcuchową o maksymalnej długości 20 znaków, w przypadku której nie jest rozróżniana wielkość liter. Nie jest to istniejący atrybut.

Dopasowywanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowywania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania, przy użyciu podłańcucha i zawiera jedną lub więcej znaków wieloznacznych.

ULW *ibm-amqDescription*
Ten atrybut LDAP zawiera opis kanału.

Ten atrybut ma pojedynczą wartość łańcuchową o maksymalnej długości 64 bajtów, bez rozróżniania wielkości liter. Nie jest to istniejący atrybut.

Dopasowywanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowywania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.

ULW *Nazwa IBM-amqConnection*
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, bez rozróżniania wielkości liter. Nie jest to istniejący atrybut.

Dopasowywanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowywania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.

ULW *Adres ibm-amqLocal*
Ten atrybut określa adres komunikacji lokalnej dla kanału.

Ten atrybut ma pojedynczą wartość łańcuchową o maksymalnej długości 48 znaków, w przypadku której nie jest rozróżniana wielkość liter. Nie jest to istniejący atrybut.

Dopasowywanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowywania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.

ULW *ibm-amqModeName*
Ten atrybut jest używany dla połączeń LU 6.2. Dodatkowa definicja parametrów sesji dla połączenia, gdy wykonywana jest alokacja sesji komunikacyjnej.

Ten atrybut ma jedną wartość łańcuchową o długości dokładnie 8 znaków, bez rozróżniania wielkości liter. Nie jest to istniejący atrybut.

Dopasowywanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowywania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.

ULW *ibm-amqPassword*
Ten atrybut LDAP określa hasło, które może być używane przez agenta MCA podczas próby zainicjowania bezpiecznej sesji LU 6.2 ze zdalnym agentem MCA.

Ten atrybut ma pojedynczą wartość całkowitą z maksymalną liczbą 12 cyfr. Nie jest to istniejący atrybut.

ULW *ibm-amqQueueManagerName*
Ten atrybut określa nazwę menedżera kolejek lub grupy menedżerów kolejek, do których aplikacja kliencka IBM MQ może zażądać połączenia.

Ten atrybut ma pojedynczą wartość łańcuchową o maksymalnej długości 48 znaków, w przypadku której nie jest rozróżniana wielkość liter. Nie jest to istniejący atrybut.

Dopasowywanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowywania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.

Odsyłacze pokrewne
“ibm-amqIsClientDefault” na stronie 1270
Ten atrybut boolowski rozwiązuje problem wyszukiwania pozycji, w których atrybut ibm-amqQueueManagerName nie został zdefiniowany.

ULW *ibm-amqSecurityExitUserData*
Ten atrybut LDAP określa dane użytkownika, które są przekazywane do wyjścia zabezpieczeń.

Ten atrybut ma pojedynczą wartość łańcuchową o maksymalnej długości 999 znaków, w przypadku której nie jest rozróżniana wielkość liter. Nie jest to istniejący atrybut.

Dopasowywanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowywania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.

ULW *ibm-amqSecurityExitName*
Ten atrybut LDAP określa nazwę programu obsługi wyjścia, który ma być uruchamiany przez wyjście zabezpieczeń kanału.

Jeśli wyjście zabezpieczeń kanału nie jest aktywne, pozostaw to pole puste.

Ten atrybut ma pojedynczą wartość łańcuchową o maksymalnej długości 999 znaków, w przypadku której nie jest rozróżniana wielkość liter. Ten atrybut nie jest wstępem do wyjścia.

Dopasowywanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowywania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.

ULW *ibm-amqSslCipherSpec*
Ten atrybut LDAP określa pojedynczą wartość atrybutu CipherSpec dla połączenia TLS.

Ten atrybut ma pojedynczą wartość łańcuchową o maksymalnej długości 32 znaków, która nie rozróżnia wielkości liter. Nie jest to istniejący atrybut.

Dopasowywanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowywania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.

ULW *ibm-amqSslPeerName*
Ten atrybut LDAP jest używany do sprawdzania nazwy wyróżniającej (DN) certyfikatu z menedżera kolejek węzła sieci lub klienta na drugim końcu kanału IBM MQ .

Ten atrybut LDAP ma pojedynczą wartość łańcuchową o maksymalnej wielkości 1024 bajtów, w przypadku której nie jest rozróżniana wielkość liter. Nie jest to już wcześniej.

Dopasowywanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowywania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.

ULW *ibm-amqTransactionProgramName*
Ten atrybut LDAP określa nazwę programu transakcyjnego. Jest on przeznaczony do użytku z połączeniami LU 6.2 .

Ten atrybut ma pojedynczą wartość łańcuchową o maksymalnej długości 64 znaków, która nie rozróżnia wielkości liter. Nie jest to już wcześniej.

Dopasowywanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowywania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.

ULW *Identyfikator ibm-amqUser*
Ten atrybut LDAP określa ID użytkownika, który ma być używany przez agenta MCA podczas próby zainicjowania bezpiecznej sesji SNA ze zdalnym agentem MCA.

Ten atrybut ma pojedynczą wartość łańcuchową o długości dokładnie 12 znaków, bez rozróżniania wielkości liter. Nie jest to istniejący atrybut.

Dopasowywanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowywania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.

ULW *ibm-amqConnectionAffinity*
Ten atrybut LDAP określa, czy aplikacje klienckie, które łączą wiele razy przy użyciu tej samej nazwy menedżera kolejek, korzystają z tego samego kanału klienta.

Ten atrybut ma pojedynczą wartość całkowitą. Nie jest to istniejący atrybut.

ULW *ibm-amqClientChannelWeight*
Ten atrybut LDAP określa wagę, która ma wpływ na używaną definicję kanału połączenia klienta.

Atrybut ważenia kanału klienta jest używany do bias wyboru definicji kanału klienta, jeśli dostępna jest więcej niż jedna odpowiednia definicja.

Ten atrybut ma pojedynczą wartość całkowitą. Nie jest to istniejący atrybut.

ULW *ibm-amqHeartBeatInterval*
Ten atrybut LDAP określa przybliżony czas między przepływami pulsu, które mają być przekazywane z wysyłającego agenta MCA, gdy w kolejce transmisji nie ma żadnych komunikatów.

Ten atrybut ma pojedynczą wartość całkowitą. Nie jest to istniejący atrybut. Wartością domyślną jest 1. Wartość domyślna jest ustawiana w bieżącej operacji zmiennej środowiskowej MQSERVER.

ULW *ibm-amqKeepAliveInterval*
Ten atrybut LDAP jest używany do określania wartości limitu czasu dla kanału.

Wartość tego atrybutu jest przekazywana do stosu komunikacyjnego określającego czas sprawdzania połączenia dla kanału. Można użyć tej opcji, aby określić inną wartość sprawdzania połączenia dla każdego kanału.

Ten atrybut ma pojedynczą wartość całkowitą. Nie jest to istniejący atrybut.

ULW *ibm-amqMaximumMessageLength*
Ten atrybut LDAP określa maksymalną długość komunikatu, który może być przesłany w kanale.

Wartością domyślną tego atrybutu jest 104857600 zgodnie z bieżącą operacją zmiennej środowiskowej MQSERVER. Ten atrybut ma pojedynczą wartość całkowitą, a nie jest to atrybut wcześniej istniejący.

ULW *Konwersacje z programem ibm-amqSharing*
Ten atrybut LDAP określa maksymalną liczbę konwersacji, które współużytkują każdą instancję kanału TCP/IP.

Ten atrybut ma pojedynczą wartość całkowitą. Ten atrybut nie jest wstępnie istniejącym atrybutem.

ULW *Typ ibm-amqTransport*
Ten atrybut LDAP określa typ transportu, który ma być używany.

Ten atrybut ma pojedynczą wartość całkowitą. Nie jest to istniejący atrybut.

ULW *ibm-amqIsClientDefault*
Ten atrybut boolowski rozwiązuje problem wyszukiwania pozycji, w których atrybut ibm-amqQueueManagerName nie został zdefiniowany.

Moduły wyjścia preconnect zwykle przeszukiwane są 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 określonego w wywołaniu MQCONN/X. Jednak w przypadku korzystania z tabel definicji kanału klienta (CCDT) można ustawić nazwę menedżera kolejek w wywołaniu MQCONN/X jako puste lub poprzedzić nazwę znakiem gwiazdki (*). Jeśli nazwa menedżera kolejek jest pusta, klient łączy się z domyślnym menedżerem kolejek. Jeśli nazwa jest poprzedzona znakiem gwiazdki (*) w menedżerze kolejek, klient łączy się z dowolnym menedżerem kolejek.

Analogicznie, atrybut ibm-amqQueueManagerName w pozycji może pozostać niezdefiniowany. W tym przypadku oczekuje się, że klient korzystający z tego punktu końcowego może połączyć się z dowolnym menedżerem kolejek. Na przykład pozycja zawiera następujące wiersze:

```
ibm-amqChannelName = "CHANNEL1"
ibm-amqConnectionName = myhost(1414)
```

W tym przykładzie klient próbuje połączyć się z określonym menedżerem kolejek działającym w systemie myhost.

Jednak w serwerach LDAP wyszukiwanie nie jest wykonywane dla wartości atrybutu, która nie została zdefiniowana. Jeśli na przykład pozycja zawiera informacje o połączeniu z wyjątkiem ibm-amqQueueManagerName, to wyniki wyszukiwania nie będą zawierały tej pozycji. Aby rozwiązać ten problem, można ustawić opcję ibm-amqIsClientDefault. Jest to atrybut boolowski i przyjmuje się, że jeśli nie jest zdefiniowany, ma wartość FALSE.

For entries where the ibm-amqQueueManagerName has not been defined and are expected to be part of the search, set ibm-amqIsClientDefault to TRUE. Jeśli jako nazwę menedżera kolejek w wywołaniu MQCONN/X określono nazwę menedżera kolejek lub gwiazdkę (*), program zewnętrzny preconnect przeszukuje serwer LDAP dla wszystkich pozycji, w których wartość atrybutu ibm-amqIsClientDefault jest ustawiona na TRUE.

Uwaga: Nie ustawiaj lub nie definiuj atrybutu ibm-amqQueueManagerName , jeśli wartość parametru ibm-amqIsClientDefault jest ustawiona na TRUE.

Odsyłacze pokrewne

“ibm-amqQueueManagerName” na stronie 1268
Ten atrybut określa nazwę menedżera kolejek lub grupy menedżerów kolejek, do których aplikacja kliencka IBM MQ może zażądać połączenia.

ULW *Kompresja ibm-amqHeader*
Ten atrybut LDAP jest listą technik kompresji danych nagłówka obsługiwanych przez kanał.

Maksymalna wielkość tego atrybutu wynosi 48 znaków. Nie jest to istniejący atrybut.

Dla tego atrybutu można podać tylko jedną wartość.

Ten atrybut listy jest określany jako łańcuchy katalogów przy użyciu formatu rozdzielonego przecinkami. Na przykład wartość określona dla parametru **ibm-amqHeaderCompression** wynosi 0 , która jest odwzorowana na wartość NONE. Wszystkie wartości, które przekraczają maksymalny dozwolony limit, są ignorowane przez klienta. Na przykład: ibm-amqHeader-kompresja zawiera maksymalnie 2 liczby całkowite na liście.

ULW *Kompresja ibm-amqMessage*
Ten atrybut LDAP jest listą technik kompresji danych komunikatu obsługiwanych przez kanał.

Maksymalna wielkość tego atrybutu wynosi 48 znaków. Nie jest to istniejący atrybut.

Ten atrybut nie obsługuje wielu wartości.

Ten atrybut listy jest określany jako łańcuchy katalogów przy użyciu formatu rozdzielonego przecinkami. Na przykład wartością określoną dla tego atrybutu jest 1,2,4, która jest odwzorowana na bazową sekwencję kompresji RLE, ZLIBFAST i ZLIBHIGH.

Wszystkie wartości, które przekraczają maksymalny dozwolony limit, są ignorowane przez klienta. Na przykład: ibm-amqMessage-kompresja zawiera maksymalnie 16 liczb całkowitych na liście.

ULW *Dane produktu IBM-amqSendExitUser*
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 przypadku której nie jest rozróżniana wielkość liter. Nie jest to istniejący atrybut.

Dopasowywanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowywania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.

Uwaga: Produkty **ibm-amqSendExitName** i **ibm-amqSendExitUserData** muszą być synchronizowane w parach. Dane użytkownika powinny być synchronizowane z nazwą wyjścia. Tak więc, jeśli ktoś jest określony, drugi również musi być określony symetrycznie, nawet jeśli nie zawiera żadnych danych.

ULW *ibm-amqSendExitName*
Ten atrybut LDAP określa nazwę programu obsługi wyjścia, który ma być uruchamiany przez wyjście wysyłania kanału.

Ten atrybut ma pojedynczą wartość łańcuchową o maksymalnej długości 999 znaków, w przypadku której nie jest rozróżniana wielkość liter. Nie jest to istniejący atrybut.

Dopasowywanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowywania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.

Uwaga: Produkty **ibm-amqSendExitName** i **ibm-amqSendExitUserData** muszą być zsynchronizowane w parach. Dane użytkownika muszą być zsynchronizowane z nazwą wyjścia. Tak więc, jeśli jest określony, drugi również musi być określony symetrycznie, nawet jeśli nie zawiera żadnych danych.

ULW *Dane programu ibm-amqReceiveExitUser*
Ten atrybut LDAP określa dane użytkownika, które są przekazywane do wyjścia odbierania.

Istnieje możliwość uruchomienia sekwencji wyjść odbierania. Łańcuch danych użytkownika dla serii wyjść jest oddzielony przecinkiem, spacjami lub dwoma znakami.

Ten atrybut ma pojedynczą wartość łańcuchową o maksymalnej długości 999 znaków, w przypadku której nie jest rozróżniana wielkość liter. Nie jest to istniejący atrybut.

Dopasowywanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowywania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.

Uwaga: Produkty **ibm-amqReceiveExitName** i **ibm-amqReceiveExitUserData** muszą być zsynchronizowane w parach. Dane użytkownika muszą być zsynchronizowane z nazwą wyjścia. Tak więc, jeśli jest określony, drugi również musi być określony symetrycznie, nawet jeśli nie zawiera żadnych danych.

ULW *ibm-amqReceiveExitName*
Ten atrybut LDAP określa nazwę programu użytkownika obsługi wyjścia, który ma być uruchamiany przez wyjście użytkownika odbierania przez kanał.

Ten atrybut jest listą nazw programów, które mają być uruchamiane w ramach sukcesji. Pozostaw puste pole, jeśli żaden kanał odbierający nie jest w stanie zakończyć działania.

Ten atrybut ma pojedynczą wartość łańcuchową o maksymalnej długości 999 znaków, w przypadku której nie jest rozróżniana wielkość liter. Nie jest to istniejący atrybut.

Dopasowywanie podłańcucha jest ignorowane. Dopasowywanie podłańcuchów jest regułą dopasowywania używaną w podschemacie, która określa zachowanie atrybutu w filtrze wyszukiwania.

Uwaga: Produkty **ibm-amqReceiveExitName** i **ibm-amqReceiveExitUserData** muszą być zsynchronizowane w parach. Dane użytkownika muszą być zsynchronizowane z nazwą wyjścia. Jeśli więc zostanie podany, drugi musi być również określony symetrycznie, nawet jeśli nie zawiera żadnych danych.

z/OS Korzystanie z przykładowych programów dla produktu z/OS

Przykładowe aplikacje proceduralne dostarczane wraz z produktem IBM MQ for z/OS demonstrują typowe zastosowania interfejsu kolejki komunikatów (Message Queue Interface-MQI).

O tym zadaniu

Produkt IBM MQ for z/OS udostępnia również przykładowe wyjścia konwersji danych, opisane w sekcji “Pisanie wyjść konwersji danych” na stronie 1082.

Wszystkie przykładowe aplikacje są dostarczane w formie kodu źródłowego; kilka jest również dostarczanych w postaci kodu wykonywalnego. Moduły źródłowe zawierają pseudocodę, która opisuje logikę programu.

Uwaga: Mimo że niektóre z przykładowych aplikacji mają podstawowe interfejsy sterowane panelem, nie mają one na celu zademonstrować, w jaki sposób zaprojektować wygląd i zachowanie aplikacji. Więcej informacji na temat projektowania interfejsów sterowanych panelami dla nieprogramowalnych terminali zawiera publikacja *SAA Common User Access: Basic Interface Design Guide* (SC26-4583) i jej dodatek (GG22-9508). Są to wytyczne pomocne przy projektowaniu aplikacji, które są spójne zarówno w aplikacji, jak i w innych aplikacjach.

Procedura

- Aby dowiedzieć się więcej na temat przykładowych programów, należy użyć poniższych odsyłaczy:
  - “Funkcje demonstrować w przykładowych aplikacjach dla produktu z/OS” na stronie 1273
  - “Przygotowywanie i uruchamianie przykładowych aplikacji dla środowiska wsadowego w systemie z/OS” na stronie 1280
  - “Przygotowywanie przykładowych aplikacji dla środowiska TSO w systemie z/OS” na stronie 1283
  - “Przygotowywanie przykładowych aplikacji dla środowiska CICS w systemie z/OS” na stronie 1285
  - “Przygotowywanie przykładowej aplikacji dla środowiska IMS w systemie z/OS” na stronie 1289
  - “Przykłady umieszczania w produkcie z/OS” na stronie 1290
  - “Pobieranie przykładów w systemie z/OS” na stronie 1292
  - “Przykład przeglądania w systemie z/OS” na stronie 1295
  - “Przykład komunikatu drukowania w systemie z/OS” na stronie 1297
  - “Przykład atrybutów kolejki w systemie z/OS” na stronie 1301
  - “Przykład programu Mail Manager w systemie z/OS” na stronie 1302
  - “Przykład kontroli kredytowej w systemie z/OS” na stronie 1309
  - “Przykład procedury obsługi komunikatów w systemie z/OS” na stronie 1321
  - “Przykład umieszczenia asynchronicznego w systemie z/OS” na stronie 1325
  - “Przykład asynchronicznego wykorzystania zadania wsadowego w systemie z/OS” na stronie 1326
  - “Przykład CICS Asynchroniczna konsumpcja i Publikowanie/subskrypcja w systemie z/OS” na stronie 1327
  - “Przykład publikowania/subskrypcji w systemie z/OS” na stronie 1330
  - “Przykład właściwości Set and Inquire message w systemie z/OS” na stronie 1332

Zadania pokrewne

“Korzystanie z przykładowych programów na platformie Multiplatforms” na stronie 1166
Te przykładowe programy proceduralne są dostarczane wraz z produktem. Przykłady są zapisywane w językach C i COBOL i demonstrują typowe zastosowania interfejsu kolejki komunikatów (Message Queue Interface-MQI).

z/OS *Funkcje demonstrować w przykładowych aplikacjach dla produktu z/OS*

Ta sekcja zawiera podsumowanie opcji MQI pokazanych w każdej przykładowej aplikacji, a także języki programowania, w których każda próbka jest pisana, oraz środowisko, w którym uruchamiane są próby.

z/OS *Umieszczanie przykładów w systemie z/OS*

Przykłady Put demonstrują sposób umieszczania komunikatów w kolejce przy użyciu wywołania MQPUT.

Aplikacja korzysta z następujących wywołań MQI:

- MQCONN
- MQOPEN
- MQPUT
- MQCLOSE

- MQDISC

Program jest dostarczany w językach COBOL i C i jest uruchamiany w środowisku wsadowym i CICS . Patrz Tabela 179 na stronie 1281 dla aplikacji wsadowej i Tabela 186 na stronie 1286 dla aplikacji CICS .

z/OS *Pobierz przykłady w systemie z/OS*

Przykłady pobierania demonstrują sposób pobierania komunikatów z kolejki przy użyciu wywołania MQGET.

Aplikacja korzysta z następujących wywołań MQI:

- MQCONN
- MQOPEN
- MQGET
- MQCLOSE
- MQDISC

Program jest dostarczany w językach COBOL i C i jest uruchamiany w środowisku wsadowym i CICS . Patrz Tabela 179 na stronie 1281 dla aplikacji wsadowej i Tabela 186 na stronie 1286 dla aplikacji CICS .

z/OS *Przeglądanie przykładu w systemie z/OS*

Przykład Przeglądaj pokazuje, jak użyć opcji Przeglądaj, aby znaleźć komunikat, wydrukować go, a następnie przejść przez komunikaty w kolejce.

Aplikacja korzysta z następujących wywołań MQI:

- MQCONN
- MQOPEN
- MQGET w celu przeglądania komunikatów
- MQCLOSE
- MQDISC

Program jest dostarczany w językach COBOL, assembler, PL/I i C. Aplikacja działa w środowisku wsadowym. Informacje na temat aplikacji wsadowej zawiera sekcja Tabela 180 na stronie 1281 .

z/OS *Przykład drukowania komunikatu w systemie z/OS*

Przykład Message Message demonstruje sposób usunięcia komunikatu z kolejki i wydrukowania danych w komunikacie, wraz ze wszystkimi polami jego deskryptora komunikatu. Może ona opcjonalnie wyświetlić wszystkie właściwości komunikatu powiązane z każdym komunikatem.

Usuwając znaki komentarza z dwóch wierszy w module źródłowym, można zmienić program w taki sposób, aby były one przeglądane, a nie usuwane, komunikaty znajdujące się w kolejce. Ten program może być używany do diagnozowania problemów z aplikacją, która umieszcza komunikaty w kolejce.

Aplikacja korzysta z następujących wywołań MQI:

- MQCONN
- MQOPEN
- MQGET w celu usunięcia komunikatów z kolejki (z opcją przeglądania)
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP

Program jest dostarczany w języku C. Aplikacja działa w środowisku wsadowym. Informacje na temat aplikacji wsadowej zawiera sekcja Tabela 181 na stronie 1282 .

z/OS *Atrybuty kolejki-przykład w systemie z/OS*

W przykładzie Atrybuty kolejki pokazano, w jaki sposób można określić i ustawić wartości atrybutów obiektu IBM MQ for z/OS .

Aplikacja korzysta z następujących wywołań MQI:

- MQOPEN
- MQINQ
- MQSET
- MQCLOSE

Program jest dostarczany w językach COBOL, assembler i C. Aplikacja działa w środowisku CICS . Informacje na ten temat zawiera sekcja Tabela 187 na stronie 1286 dla aplikacji CICS .

z/OS *Przykład programu Mail Manager w systemie z/OS*

Uwagi dotyczące używania przykładu menedżera poczty elektronicznej.

Przykład menedżera poczty elektronicznej demonstruje następujące techniki:

- Korzystanie z kolejek aliasowych
- Korzystanie z kolejki modelowej w celu utworzenia tymczasowej kolejki dynamicznej
- Korzystanie z kolejek odpowiedzi
- Korzystanie z punktów synchronizacji w środowisku CICS i w środowiskach wsadowych
- Wysyłanie komend do systemowej kolejki wejściowej komend
- Testowanie kodów powrotu
- Wysyłanie komunikatów do menedżerów kolejek zdalnych, zarówno przy użyciu lokalnej definicji kolejki zdalnej, jak i poprzez umieszczanie komunikatów bezpośrednio w kolejce o określonej nazwie w zdalnym menedżerze kolejek

Aplikacja korzysta z następujących wywołań MQI:

- MQCONN
- 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 korzystają z adaptera wsadowego IBM MQ for z/OS i zawierają niektóre panele ISPF.

Więcej informacji na ten temat zawiera sekcja Tabela 184 na stronie 1283 dla aplikacji TSO i Tabela 188 na stronie 1287 w przypadku aplikacji CICS .

z/OS *Przykład kontroli kredytowej w systemie z/OS*

Informacje te zawierają punkty, które należy wziąć pod uwagę podczas korzystania z przykładu kontroli kredytowej.

Przykład kontroli kredytowej to pakiet programów, który demonstruje te techniki:

- Projektowanie aplikacji, która działa w więcej niż jednym środowisku
- Korzystanie z kolejki modelowej w celu utworzenia tymczasowej kolejki dynamicznej
- Korzystanie z identyfikatora korelacji
- Ustawianie i przekazywanie informacji kontekstowych
- Korzystanie z priorytetu i trwałości komunikatów
- 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 korzysta z następujących wywołań MQI:

- MQOPEN
- MQPUT
- MQPUT1
- MQGET służy do przeglądania i pobierania komunikatów, korzystania z opcji oczekiwania i sygnału, a także do pobierania konkretnego komunikatu
- MQINQ
- MQSET
- MQCLOSE

Przykład może być uruchamiany jako autonomiczna aplikacja produktu CICS . Jednak w celu zademonstrowania sposobu zaprojektowania aplikacji kolejkowania komunikatów, która korzysta z urządzeń udostępnianych zarówno przez środowiska CICS , jak i IMS , jeden moduł jest również dostarczany jako program przetwarzania komunikatów wsadowych IMS .

Programy CICS są dostarczane w językach C i COBOL. Pojedynczy program IMS jest dostarczany w języku C.

Informacje na ten temat zawiera sekcja Tabela 189 na stronie 1287 dla aplikacji CICS i Tabela 191 na stronie 1289 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 korzysta z następujących wywołań MQI:

- MQCONN
- MQOPEN
- MQINQ
- MQPUT1
- MQCMIT
- MQBACK
- MQGET
- MQCLOSE
- MQDISC

Program jest dostarczany w językach programowania C i COBOL. Aplikacja jest uruchamiana w TSO. PatrzTabela 185 na stronie 1284dla aplikacji TSO.

z/OS *Przykłady rozproszonego wyjścia kolejkowania w systemie z/OS*

Tabela programów źródłowych w próbkach wyjścia rozproszonego kolejkowania.

Nazwy programów źródłowych w próbkach wyjścia rozproszonego kolejkowania są wymienione w poniższej tabeli:

Tabela 177. Kod źródłowy dla przykładów rozproszonego wyjścia kolejkowania

Nazwa elementu klastra	**Dla języka**	**Opis**	**Dostarczane w bibliotece**
CSQ4BAX0	Asembler	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 odsyłaczy z kodem CSQXSTUB.

z/OS *Przykłady wyjścia konwersji danych w systemie z/OS*

Szkielet jest dostarczany dla procedury wyjścia konwersji danych, a przykład jest dostarczany razem z programem IBM MQ ilustrującym wywołanie MQXCNVC.

Nazwy programów źródłowych przykładowych danych wyjściowych konwersji danych są wymienione w poniższej tabeli:

Tabela 178. Źródło dla przykładów wyjścia konwersji danych (tylko język asembler)

Nazwa elementu klastra	**Opis**	**Dostarczane w bibliotece**
CSQ4BAX8	Program źródłowy	SCSQASMS
CSQ4BAX9	Program źródłowy	SCSQASMS
CSQ4CAX9	Program źródłowy	SCSQASMS

Uwaga: Programy źródłowe są edytowane pod odsyłaczem CSQASTUB.

Więcej informacji zawiera sekcja “Pisanie wyjść konwersji danych” na stronie 1082.

z/OS *Przykłady publikowania/subskrypcji w systemie z/OS*

Przykładowe programy publikowania/subskrypcji demonstrują sposób użycia funkcji publikowania i subskrypcji w produkcie IBM MQ.

Istnieją cztery programy przykładowe języka programowania w języku C i dwóch językach COBOL demonstruje sposób programu do interfejsu publikowania/subskrybowania produktu IBM MQ .

Aplikacje korzystają z następujących wywołań MQI:

- MQCONN
- MQOPEN
- MQPUT
- MQSUB
- MQGET
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH

- MQINQMP

Programy przykładowe publiczne/subskrybowane są dostarczane w językach programowania w językach C i COBOL. Przykładowe aplikacje działają w środowisku wsadowym. Więcej informacji na ten temat zawiera sekcja Przykłady publikowania/subskrypcji dla aplikacji wsadowych.

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 akceptować 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, wydając komendę MQSC:

```
DISPLAY QMGR CHLAUTH
```

Ważne: To zadanie oczekuje, że włączone są rekordy uwierzytelniania kanału. Jeśli jest to menedżer kolejek używany przez innych użytkowników i aplikacje, zmiana tego ustawienia będzie mieć 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 wyjście bezpieczeństwa), które ustawia wartość MCAUSER na *identyfikator-użytkownika bez uprawnień uprzywilejowanych* , który zostanie uzyskany w kroku “1” na stronie 1278.

Należy wiedzieć, która nazwa kanału ma być używana przez aplikację w taki sposób, aby aplikacja mogła używać kanału. Należy również wiedzieć, które obiekty, na przykład kolejki lub tematy, mają być używane przez aplikację, dzięki czemu aplikacja będzie mogła je używać.

O tym zadaniu

To zadanie tworzy nieuprawnionego ID użytkownika, który ma być używany dla aplikacji klienckiej, która łączy się z menedżerem kolejek. Dostęp do aplikacji klienckiej jest udzielany tylko w celu użycia kanału, którego potrzebuje, oraz kolejki, której potrzebuje, korzystając z tego identyfikatora użytkownika.

Procedura

1. Uzyskaj identyfikator użytkownika w systemie, na którym jest uruchomiony menedżer kolejek.

   W przypadku tego zadania ten identyfikator użytkownika nie może być uprzywilejowanym użytkownikiem administracyjnym. Ten ID użytkownika to uprawnienie, w ramach którego połączenie klienta zostanie uruchomione w menedżerze kolejek.
2. Uruchom program nasłuchujący.
   a) Upewnij się, że inicjator kanału jest uruchomiony. Jeśli nie, uruchom go, wydając komendę **START CHINIT** .
   b) Uruchom program nasłuchujący, wydając następującą komendę:

   ```
   START LISTENER TRPTYPE(TCP) PORT(nnnn)
   ```

   gdzie *nnnn* jest wybranym numerem portu.
3. Jeśli aplikacja korzysta z systemu SYSTEM.DEF.SVRCONN , a następnie ten kanał jest już zdefiniowany. Jeśli aplikacja używa innego kanału, utwórz ją, wydając 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, zezwalając na korzystanie z kanału tylko przez adres IP systemu klienckiego za pomocą komendy MQSC:

```
SET CHLAUTH(' channel-name ') TYPE(ADDRESSMAP) ADDRESS(' client-machine-IP-address ') +
MCAUSER(' non-privileged-user-id ')
```

   where

   nazwa-kanału jest nazwą kanału.
   adres_IP jest adresem IP systemu klienckiego użytkownika. Jeśli przykładowa aplikacja kliencka działa na tym samym komputerze, co menedżer kolejek, należy użyć adresu IP '127.0.0.1', jeśli aplikacja ma nawiązać połączenie przy użyciu 'localhost'. Jeśli zostanie nawiązane połączenie kilku różnych komputerów klienckich, można użyć wzorca lub zakresu zamiast pojedynczego adresu IP. Szczegółowe informacje na ten temat zawiera sekcja Ogólne adresy IP .
   identyfikator-użytkownika bez uprawnień jest identyfikatorem użytkownika uzyskanym w kroku “1” na stronie 1278
5. Jeśli aplikacja korzysta z systemu SYSTEM.DEFAULT.LOCAL.QUEUE, a następnie ta kolejka jest już zdefiniowana. Jeśli aplikacja używa innej kolejki, utwórz ją, wydając komendę 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 sprawdź, czy menedżer kolejek:
   a) Upewnij się, że inicjator kanału jest uruchomiony. Jeśli nie, uruchom inicjator 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, że korzysta z kolejek, nadając uprawnienia do uzyskiwania informacji oraz umieszczając i pobierając komunikaty przy użyciu kolejki przy użyciu identyfikatora użytkownika, który ma być używany, wydając komendy MQSC:
   Wydaj komendę 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-użytkownika bez uprawnień jest identyfikatorem użytkownika uzyskanym w kroku “1” na stronie 1278
8. Jeśli aplikacja jest aplikacją publikowania/subskrybowania, to znaczy, że korzysta z tematów, przyznaj dostęp, aby umożliwić publikowanie i subskrybowanie tematu przy użyciu identyfikatora użytkownika, który ma być używany, wydając następujące komendy produktu 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-użytkownika bez uprawnień jest identyfikatorem użytkownika uzyskanym w kroku “1” na stronie 1278
Spowoduje to nadanie *nieuprawnionego identyfikatora użytkownika* dostępu do dowolnego tematu w drzewie tematów. Można również zdefiniować obiekt tematu przy użyciu produktu **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 połączyć się z menedżerem kolejek i umieścić lub pobrać komunikaty przy użyciu kolejki.

Pojęcia pokrewne

z/OS Uprawnienia do pracy z obiektami IBM MQ w systemie z/OS

Odsyłacze pokrewne

USTAW WARTOŚĆ CHLAUTH
Zdefiniowanie kanału
DEFINE QLOCAL
SET AUTHREC

z/OS *Przygotowywanie i uruchamianie przykładowych aplikacji dla środowiska wsadowego w systemie z/OS*

Aby przygotować przykładową aplikację, która jest uruchamiana w środowisku wsadowym, należy wykonać te same kroki, które podczas budowania dowolnej aplikacji wsadowej IBM MQ for z/OS .

Te kroki są wymienione w sekcji “Budowanie aplikacji wsadowych produktu z/OS” na stronie 1130.

Alternatywnie, w przypadku, gdy dostarczamy kod wykonywalny przykładu, można go uruchomić z biblioteki ładowania thlqual.SCSQLOAD .

Uwaga: Wersja językowa programu asemblera w przykładzie przeglądania używa bloków sterujących danych (DCBs), więc należy je połączyć za pomocą programu `RMODE(24)`.

Elementy biblioteki, które mają być używane, są wymienione w następujących elementach: Tabela 179 na stronie 1281, Tabela 180 na stronie 1281, Tabela 181 na stronie 1282i Tabela 182 na stronie 1282.

Należy zmodyfikować uruchomienie JCL dostarczonego dla przykładów, które mają być używane (patrz Tabela 179 na stronie 1281, Tabela 180 na stronie 1281, Tabela 181 na stronie 1282i Tabela 182 na stronie 1282).

Instrukcja PARM w dostarczonym JCL zawiera pewną liczbę parametrów, które należy zmodyfikować. Aby uruchomić programy przykładowe w języku C, rozdziel parametry za pomocą spacji; aby uruchomić programy przykładowe asembler, COBOL i PL/I, rozdziel je przecinkami. Na przykład, jeśli nazwą menedżera kolejek jest CSQ1 , a użytkownik chce uruchomić aplikację z kolejką o nazwie LOCALQ1, w języku COBOL, PL/I i assembler-language JCL, instrukcja PARM powinna wyglądać następująco:

```
PARM=(CSQ1,LOCALQ1)
```

W języku JCL języka C instrukcja PARM powinna wyglądać następująco:

```
PARM=('CSQ1 LOCALQ1')
```

Teraz jesteś gotowy do wprowadzenia zadań.

z/OS *Nazwy przykładowych aplikacji wsadowych w systemie z/OS*

Podsumowanie programów, które są dostarczane dla przykładowych aplikacji wsadowych.

Programy aplikacji wsadowych są podsumowane w następujących tabelach:

- Przykłady Tabela 179 na stronie 1281 Put i Get
- Tabela 180 na stronie 1281 Przeglądaj przykład
- Tabela 181 na stronie 1282 Przykład drukowania komunikatu
- Tabela 182 na stronie 1282 Przykłady publikowania/subskrypcji
- Tabela 183 na stronie 1283 Inne przykłady

Tabela 179. Przykłady umieszczania i pobierania zadania wsadowego

Nazwa elementu klastra	Dla języka	Opis	Zbiór źródłowy dostarczony w bibliotece	Plik wykonywalny dostarczany 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	COBOL	Pobierz program źródłowy	SCSQCOBS	SCSQLOAD
CSQ4BVK1	COBOL	Umieść program źródłowy	SCSQCOBS	SCSQLOAD
CSQ4BVJR	COBOL	Przykładowe uruchomienie JCL dla CSQBVJ1 i CSQBVK1	SCSQPROC	Brak

Tabela 180. Przykład przeglądania wsadowego

Nazwa elementu klastra	Dla języka	Opis	Zbiór źródłowy dostarczony w bibliotece	Plik wykonywalny dostarczany w bibliotece
CSQ4BVA1	COBOL	Program źródłowy	SCSQCOBS	SCSQLOAD
CSQ4BVAR	COBOL	Przykładowe uruchomienie JCL dla CSQ4BVA1	SCSQPROC	Brak
CSQ4BAA1	Asembler	Program źródłowy	SCSQASMS	SCSQLOAD
CSQ4BAAR	Asembler	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

Tabela 180. Przykład przeglądania wsadowego (kontynuacja)

Nazwa elementu klastra	Dla języka	Opis	Zbiór źródłowy dostarczony w bibliotece	Plik wykonywalny dostarczany w bibliotece
CSQ4BPAR	PL/I	Przykładowe uruchomienie JCL dla CSQ4BPA1	SCSQPROC	Brak

Tabela 181. Próbka komunikatu drukowania wsadowego (tylko w języku C)

Nazwa elementu klastra	Opis	Zbiór źródłowy dostarczony w bibliotece	Plik wykonywalny dostarczany 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 182. Przykłady publikowania/subskrypcji

Nazwa elementu klastra	Dla języka	Opis	Zbiór źródłowy dostarczony w bibliotece	JCL w SCSQPROC	Plik wykonywalny dostarczany w bibliotece
CSQ4BCP1	C	Publikuj w programie źródłowym tematów	SCSQC37S	CSQ4BCPP	SCSQLOAD
CSQ4BCP2	C	Subskrybuj temat i pobierz program źródłowy wiadomości	SCSQC37S	CSQ4BCPS	SCSQLOAD
CSQ4BCP3	C	Subskrybuj temat przy użyciu podanego przez użytkownika miejsca docelowego i pobierz program źródłowy wiadomości	SCSQC37S	CSQ4BCPD	SCSQLOAD
CSQ4BCP4	C	Subskrybuj temat przy użyciu opcji rozszerzonych i pobierz program źródłowy wiadomości	SCSQC37S	CSQ4BCPE	SCSQLOAD
CSQ4BVP1	COBOL	Publikuj w programie źródłowym tematów	SCSQCOBS	CSQ4BVPP	SCSQLOAD
CSQ4BVP2	COBOL	Subskrybuj temat i pobierz program źródłowy wiadomości	SCSQCOBS	CSQ4BVPS	SCSQLOAD

Tabela 183. Pozostałe próbki

Nazwa elementu klastra	Dla języka	Opis	Zbiór źródłowy dostarczony w bibliotece	JCL w SCSQPROC	Plik wykonywalny dostarczany w bibliotece
CSQ4BCS1	C	Program źródłowy wykorzystania asynchronicznego	SCSQC37S	CSQ4BCSC	SCSQLOAD
CSQ4BCS2	C	Program źródłowy asynchronicznego umieszczania i sprawdzania statusu	SCSQC37S	CSQ4BCSP	SCSQLOAD
CSQ4BCM1	C	Zapytanie o 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, które należy wykonać podczas budowania dowolnej aplikacji wsadowej IBM MQ for z/OS .

Te kroki są wymienione w sekcji “Budowanie aplikacji wsadowych produktu z/OS” na stronie 1130. Elementy biblioteki, które mają być używane, są wymienione w sekcji Tabela 184 na stronie 1283.

Alternatywnie, w przypadku, gdy dostarczamy kod wykonywalny przykładu, można go uruchomić z biblioteki ładowania thlqual.SCSQLOAD .

W przypadku przykładowej aplikacji Mail Manager upewnij się, że kolejki używane przez nią są dostępne w systemie. Są one zdefiniowane w elemencie **thlqual**.SCSQPROC (CSQ4CVD). Aby upewnić się, że kolejki te są zawsze dostępne, można dodać te elementy do zestawu danych wejściowych inicjowania CSQINP2 lub użyć programu CSQUTIL w celu załadowania tych definicji kolejek.

z/OS *Nazwy przykładowych aplikacji TSO w systemie z/OS*

Informacje na temat nazw programów dostarczanych dla każdej z przykładowych aplikacji TSO oraz bibliotek, w których znajduje się kod źródłowy, JCL i, tylko dla przykładu procedury obsługi komunikatów, pliki wykonywalne.

Programy aplikacji TSO są podsumowane w następujących tabelach:

- Tabela 184 na stronie 1283 Przykład menedżera poczty
- Tabela 185 na stronie 1284 Przykład procedury obsługi komunikatów

Te przykłady wykorzystują panele ISPF. W związku z tym należy dołączyć kod pośredniczący ISPF, ISPLINK, podczas tworzenia odsyłaczy do edycji programów.

Tabela 184. Przykład programu TSO Mail Manager

Nazwa elementu klastra	Dla języka	Opis	Zbiór źródłowy dostarczony w bibliotece
CSQ4CVD	Niezależne	Definicje obiektów produktu IBM MQ for z/OS	SCSQPROC

Tabela 184. Przykład programu TSO Mail Manager (kontynuacja)

Nazwa elementu klastra	Dla języka	Opis	Zbiór źródłowy dostarczony w bibliotece
CSQ40	Niezależne	Komunikaty protokołu ISPF	SCSQMSGE
CSQ4RVD1	COBOL	CLIST, aby zainicjować CSQ4TVD1	SCSQCLST
CSQ4TVD1	COBOL	Program źródłowy dla programu Menu	SCSQCOBS
CSQ4TVD2	COBOL	Program źródłowy programu Get Mail	SCSQCOBS
CSQ4TVD4	COBOL	Program źródłowy dla programu wysyłania poczty	SCSQCOBS
CSQ4TVD5	COBOL	Program źródłowy dla programu Nickname	SCSQCOBS
CSQ4VDP1-6	COBOL	Definicje paneli	SCSQPNLA
CSQ4VD0	COBOL	Definicja danych	SCSQCOBC
CSQ4VD1	COBOL	Definicja danych	SCSQCOBC
CSQ4VD2	COBOL	Definicja danych	SCSQCOBC
CSQ4VD4	COBOL	Definicja danych	SCSQCOBC
CSQ4RCD1	C	CLIST, aby zainicjować CSQ4TCD1	SCSQCLST
CSQ4TCD1	C	Program źródłowy dla programu Menu	SCSQC37S
CSQ4TCD2	C	Program źródłowy programu Get Mail	SCSQC37S
CSQ4TCD4	C	Program źródłowy dla programu wysyłania poczty	SCSQC37S
CSQ4TCD5	C	Program źródłowy dla programu Nickname	SCSQC37S
CSQ4CDP1-6	C	Definicje paneli	SCSQPNLA
CSQ4TC0	C	Plik włączany	SCSQC370

Tabela 185. Przykład procedury obsługi komunikatów TSO

Nazwa elementu klastra	Dla języka	Opis	Zbiór źródłowy dostarczony w bibliotece	Plik wykonywalny dostarczany w bibliotece
CSQ4TCH0	C	Definicja danych	SCSQC370	Brak
CSQ4TCH1	C	Program źródłowy	SCSQC37S	SCSQLOAD
CSQ4TCH2	C	Program źródłowy	SCSQC37S	SCSQLOAD

Tabela 185. Przykład procedury obsługi komunikatów TSO (kontynuacja)				
Nazwa elementu klastra	**Dla języka**	**Opis**	**Zbiór źródłowy dostarczony w bibliotece**	**Plik wykonywalny dostarczany w bibliotece**
CSQ4TCH3	C	Program źródłowy	SCSQC37S	SCSQLOAD
CSQ4RCH1	C i COBOL	CLIST do 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	COBOL	Definicja danych	SCSQCOBC	Brak
CSQ4TVH1	COBOL	Program źródłowy	SCSQCOBS	SCSQLOAD
CSQ4TVH2	COBOL	Program źródłowy	SCSQCOBS	SCSQLOAD
CSQ4TVH3	COBOL	Program źródłowy	SCSQCOBS	SCSQLOAD

z/OS *Przygotowywanie przykładowych aplikacji dla środowiska CICS w systemie z/OS*

Przed uruchomieniem programów przykładowych produktu CICS należy zalogować się do produktu CICS przy użyciu parametru LOGMODE o wartości 32702. Wynika to z faktu, że programy przykładowe zostały napisane w celu użycia ekranu trybu 3270 2.

Aby przygotować przykładową aplikację, która działa w środowisku CICS , wykonaj następujące kroki:

1. Utwórz odwzorowanie opisu symbolicznego i odwzorowanie ekranu fizycznego dla przykładu, składając źródło definicji ekranu BMS (dostarczone w bibliotece **thlqual**.SCSQMAPS, gdzie **thlqual** jest kwalifikatorem wysokiego poziomu używanym przez instalację). Po nazwie odwzorowania należy użyć nazwy źródła definicji ekranu BMS (nie są dostępne dla programów umieszczania i pobierania próbek), ale należy pominąć ostatni znak tej nazwy.
2. Wykonaj te same kroki, które podczas budowania dowolnej aplikacji CICS IBM MQ for z/OS . Te kroki są wymienione w sekcji “Budowanie aplikacji CICS w produkcie z/OS” na stronie 1133. Elementy biblioteki, które mają być używane, są wymienione w następujących elementach: Tabela 186 na stronie 1286, Tabela 187 na stronie 1286, Tabela 188 na stronie 1287i Tabela 189 na stronie 1287.

   Alternatywnie, jeśli dostarczamy kod wykonywalny przykładu, można go uruchomić z biblioteki ładowania systemu thlqual.SCSQCICS .
3. Zidentyfikuj zestaw map, programy i transakcję do programu CICS , aktualizując zestaw danych definicji systemu CICS (CSD). Wymagane definicje znajdują się w elemencie **thlqual**.SCSQPROC (CSQ4S100). Wytyczne dotyczące tego sposobu można znaleźć w sekcji *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 aplikacji przykładowej kontroli kredytowej wyświetlany jest komunikat o błędzie na tym etapie, jeśli nie utworzono jeszcze zestawu danych VSAM, którego użyto do próby.
4. W przypadku przykładowych aplikacji Check Check i Mail Manager upewnij się, że kolejki, których używają, są dostępne w systemie. Dla przykładu Sprawdzenie uznania są one zdefiniowane w podzbiorze **thlqual**.SCSQPROC (CSQ4CVB) dla języka COBOL oraz **thlqual**.SCSQPROC (CSQ4CCB) dla C. Dla przykładu menedżera poczty elektronicznej są one zdefiniowane w elemencie **thlqual**.SCSQPROC (CSQ4CVD). Aby upewnić się, że kolejki te są zawsze dostępne, można dodać te

elementy do zestawu danych wejściowych inicjowania CSQINP2 lub użyć programu CSQUTIL w celu załadowania tych definicji kolejek.

W przykładowej aplikacji Atrybuty kolejki można użyć jednej lub większej liczby kolejek dostarczonych dla innych aplikacji przykładowych. Alternatywnie można użyć własnych kolejek. Jednak w postaci, w której jest ona dostarczana, ten przykład działa tylko z kolejkami, których nazwy zawierają znaki CSQ4SAMP (w pierwszych ośmiu bajtach).

z/OS *Nazwy przykładowych aplikacji produktu CICS w systemie z/OS*

Ten temat zawiera podsumowanie programów dostarczanych dla przykładowych aplikacji produktu CICS .

Programy aplikacji CICS są podsumowane w następujących tabelach:

- Przykłady Tabela 186 na stronie 1286 Put i Get
- Tabela 187 na stronie 1286 Przykład atrybutów kolejki
- Tabela 188 na stronie 1287 Przykład programu Mail Manager (tylko w języku COBOL)
- Tabela 189 na stronie 1287 Przykład kontroli kredytowej
- Tabela 190 na stronie 1288 Przykłady wykorzystania asynchronicznego i publikowania/subskrypcji

Tabela 186. Przykłady CICS Put i Get

Nazwa elementu klastra	Dla języka	Opis	Zbiór źródłowy dostarczony w bibliotece	Plik wykonywalny dostarczany w bibliotece
CSQ4CCK1	C	Umieść program źródłowy	SCSQC37S	SCSQCICS
CSQ4CCJ1	C	Pobierz program źródłowy	SCSQC37S	SCSQCICS
CSQ4CVJ1	COBOL	Pobierz program źródłowy	SCSQCOBS	SCSQCICS
CSQ4CVK1	COBOL	Umieść program źródłowy	SCSQCOBS	SCSQCICS
CSQ4S100	Niezależne	Zestaw danych definicji systemu CICS	SCSQPROC	Brak

Tabela 187. CICS Przykład atrybutów kolejki

Nazwa elementu klastra	Dla języka	Opis	Zbiór źródłowy dostarczony w bibliotece	Plik wykonywalny dostarczany w bibliotece
CSQ4CVC1	COBOL	Program źródłowy	SCSQCOBS	SCSQCICS
CSQ4VMSG	COBOL	Definicja komunikatu	SCSQCOBC	Brak
CSQ4VCMS	COBOL	Definicja ekranu BMS	SCSQMAPS	SCSQCICS (o nazwie CSQ4ACM)
CSQ4CAC1	Asembler	Program źródłowy	SCSQASMS	SCSQCICS
CSQ4AMSG	Asembler	Definicja komunikatu	SCSQMACS	Brak
CSQ4ACMS	Asembler	Definicja ekranu BMS	SCSQMAPS	SCSQCICS (o nazwie CSQ4ACM)

Tabela 187. CICS Przykład atrybutów kolejki (kontynuacja)

Nazwa elementu klastra	Dla języka	Opis	Zbiór źródłowy dostarczony w bibliotece	Plik wykonywalny dostarczany w bibliotece
CSQ4CCC1	C	Program źródłowy	SCSQC37S	SCSQCICS
CSQ4CMSG	C	Definicja komunikatu	SCSQC370	Brak
CSQ4CCMS	C	Definicja ekranu BMS	SCSQMAPS	SCSQCICS (o nazwie CSQ4ACM)
CSQ4S100	Niezależne	Zestaw danych definicji systemu CICS	SCSQPROC	Brak

Tabela 188. CICS Przykład programu Mail Manager (tylko w języku COBOL)

Nazwa elementu klastra	Opis	Zbiór źródłowy dostarczony w bibliotece
CSQ4CVD	Definicje obiektów produktu IBM MQ for z/OS	SCSQPROC
CSQ4CVD1	Źródło dla programu Menu	SCSQCOBS
CSQ4CVD2	Źródło programu Get Mail	SCSQCOBS
CSQ4CVD3	Źródło dla programu wyświetlania komunikatu	SCSQCOBS
CSQ4CVD4	Źródło dla programu Wyślij pocztę	SCSQCOBS
CSQ4CVD5	Źródło dla programu Nickname	SCSQCOBS
CSQ4VDMS	Źródło definicji ekranu BMS	SCSQMAPS
CSQ4S100	Zestaw danych definicji systemu CICS	SCSQPROC
CSQ4VD0	Definicja danych	SCSQCOBC
CSQ4VD3	Definicja danych	SCSQCOBC
CSQ4VD4	Definicja danych	SCSQCOBC

Tabela 189. CICS Przykład kontroli kredytowej

Nazwa elementu klastra	Dla języka	Opis	Zbiór źródłowy dostarczony w bibliotece
CSQ4CVB	Niezależne	Definicje obiektów produktu IBM MQ	SCSQPROC
CSQ4CCB	Niezależne	Definicje obiektów produktu IBM MQ	SCSQPROC
CSQ4CVB1	COBOL	Źródło dla programu user-interface	SCSQCOBS
CSQ4CVB2	COBOL	Źródło dla menedżera aplikacji kredytowych	SCSQCOBS
CSQ4CVB3	COBOL	Źródło dla programu checking-account	SCSQCOBS

Tabela 189. CICS Przykład kontroli kredytowej (kontynuacja)

Nazwa elementu klastra	Dla języka	Opis	Zbiór źródłowy dostarczony w bibliotece
CSQ4CVB4	COBOL	Źródło dla programu dystrybucyjnego	SCSQCOBS
CSQ4CVB5	COBOL	Źródło dla programu query-query	SCSQCOBS
CSQ4CCB1	C	Źródło dla programu user-interface	SCSQC37S
CSQ4CCB2	C	Źródło dla menedżera aplikacji kredytowych	SCSQC37S
CSQ4CCB3	C	Źródło dla programu checking-account	SCSQC37S
CSQ4CCB4	C	Źródło dla programu dystrybucyjnego	SCSQC37S
CSQ4CCB5	C	Źródło dla programu query-query	SCSQC37S
CSQ4CB0	C	Plik włączany	SCSQC370
CSQ4CBMS	C	Źródło definicji ekranu BMS	SCSQMAPS
CSQ4VBMS	COBOL	Źródło definicji ekranu BMS	SCSQMAPS
CSQ4VB0	COBOL	Definicja danych	SCSQCOBC
CSQ4VB1	COBOL	Definicja danych	SCSQCOBC
CSQ4VB2	COBOL	Definicja danych	SCSQCOBC
CSQ4VB3	COBOL	Definicja danych	SCSQCOBC
CSQ4VB4	COBOL	Definicja danych	SCSQCOBC
CSQ4VB5	COBOL	Definicja danych	SCSQCOBC
CSQ4VB6	COBOL	Definicja danych	SCSQCOBC
CSQ4VB7	COBOL	Definicja danych	SCSQCOBC
CSQ4VB8	COBOL	Definicja danych	SCSQCOBC
CSQ4BAQ	Niezależne	Źródło dla zestawu danych VSAM	SCSQPROC
CSQ4FILE	Niezależne	Kod JCL w celu zbudowania zestawu danych VSAM używanego przez CSQ4CVB3	SCSQPROC
CSQ4S100	Niezależne	Zestaw danych definicji systemu CICS	SCSQPROC

Tabela 190. CICS Przykłady wykorzystania asynchronicznego i publikowania/subskrypcji

Nazwa elementu klastra	Opis	Zbiór źródłowy dostarczony w bibliotece
CSQ4CVCN	Źródło dla programu Simple Message Consumption	SCSQCOBS
CSQ4CVCT	Źródło dla programu kontroli wykorzystania komunikatów	SCSQCOBS
CSQ4CVEV	Kod źródłowy programu obsługi zdarzeń	SCSQCOBS

Tabela 190. CICS Przykłady wykorzystania asynchronicznego i publikowania/subskrypcji (kontynuacja)		
Nazwa elementu klastra	**Opis**	**Zbiór źródłowy dostarczony w bibliotece**
CSQ4CVPT	Źródło dla programu klienckiego typu Message Put	SCSQCOBS
CSQ4CVRG	Kod źródłowy dla programu klienckiego rejestracji	SCSQCOBS
CSQ4S100	CICS Zestaw danych definicji systemu	SCSQPROC

z/OS *Przygotowywanie przykładowej aplikacji dla środowiska IMS w systemie z/OS*

Część przykładowej aplikacji Sprawdzenie kredytu może być uruchamiana w środowisku IMS .

Aby przygotować tę część aplikacji do uruchamiania z przykładową CICS , wykonaj kroki opisane w sekcji “Przygotowywanie przykładowych aplikacji dla środowiska CICS w systemie z/OS” na stronie 1285.

Następnie należy wykonać następujące kroki:

1. Wykonaj te same kroki, które podczas budowania dowolnej aplikacji IMS IBM MQ for z/OS . Te kroki są wymienione w sekcji “Budowanie aplikacji IMS (BMP lub MPP)” na stronie 1134. Elementy biblioteki, które mają być używane, są wymienione w sekcji Tabela 191 na stronie 1289.
2. Zidentyfikuj program użytkowy i bazę danych do programu IMS. Przykłady są dostarczane wraz z instrukcjami PSBGEN, DBDGEN, ACB, IMSGEN i IMSDALOC w celu umożliwienia tego działania.
3. Załaduj bazę danych CSQ4CA , dostosowując i uruchamiając przykładowe JCL udostępnione dla tego celu (CSQ4ILDB). Ten skrypt JCL ładuje bazę danych z danymi 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 (BMP), dostosowując i uruchamiając przykładowe JCL udostępnione w tym celu. Ten JCL uruchamia program BMP zorientowany na zadania wsadowe. Aby uruchomić program jako program BMP zorientowany na komunikaty, należy usunąć 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 plików JCL, które są dostarczane w przykładowej aplikacji IMS .

Tabela 191. Kod źródłowy i kod JCL dla przykładu Sprawdzenie kredytu IMS (tylko w języku C)		
Nazwa elementu klastra	**Opis**	**Dostarczane w bibliotece**
CSQ4CVB	Definicje obiektów produktu IBM MQ	SCSQPROC
CSQ4ICB3	Źródło dla programu checking-account	SCSQC37S
CSQ4ICBL	Źródło ładowania bazy danych kont kontrolnych	SCSQC37S
CSQ4CBI	Definicja danych	SCSQC370
CSQ4PSBL	PSBGEN JCL dla programu ładowanego przez bazę danych	SCSQPROC
CSQ4PSB3	PSBGEN JCL dla programu checking-account	SCSQPROC

Tabela 191. Kod źródłowy i kod JCL dla przykładu Sprawdzenie kredytu IMS (tylko w języku C) (kontynuacja)		
Nazwa elementu klastra	**Opis**	**Dostarczane w bibliotece**
CSQ4DBDS	DBDGEN JCL dla bazy danych CSQ4CA	SCSQPROC
CSQ4GIMS	Definicje makr produktu IMSGEN dla CSQ4IVB3 i CSQ4CA	SCSQPROC
CSQ4ACBG	Definicja bloku kontrolnego aplikacji (Application Control Block-ACB) dla CSQ4IVB3	SCSQPROC
CSQ4BAQ	Źródło dla bazy danych	SCSQPROC
CSQ4ILDB	Przykładowe uruchomienie JCL dla zadania ładowania bazy danych	SCSQPROC
CSQ4ICBR	Przykładowy skrypt uruchamiający JCL dla programu do sprawdzania kont	SCSQPROC
CSQ4DYNA	IMSDefinicje makr DALOC dla bazy danych	SCSQPROC

z/OS *Przykłady umieszczania w produkcie z/OS*

Programy przykładowe umieszczone w kolejce umieszczają komunikaty w kolejce przy użyciu wywołania MQPUT.

Programy źródłowe są dostarczane w językach C i COBOL w środowiskach wsadowych i CICS (patrz Tabela 179 na stronie 1281 i Tabela 186 na stronie 1286).

Wzór próbki umieszczonej

Przepływ przez logikę programu jest następujący:

1. Połącz się z menedżerem kolejek przy użyciu wywołania MQCONN. Jeśli wywołanie nie powiedzie się, wydrukuj kody zakończenia i przyczyny, a następnie zatrzymaj przetwarzanie.

   Uwaga: Jeśli próbka jest uruchamiana w środowisku produktu CICS , nie ma potrzeby wydawania wywołania MQCONN. Jeśli zostanie to wykonalne, zwraca wartość DEF_HCONN. Za pomocą uchwytu połączenia MQHC_DEF_HCONN można korzystać w przypadku wywołań MQI.
2. Otwórz kolejkę za pomocą wywołania MQOPEN z opcją MQOO_OUTPUT. Po wejściu do tego wywołania program korzysta z uchwytu połączenia, który jest zwracany w kroku “1” na stronie 1292. W przypadku struktury deskryptora obiektu (MQOD) używa ona wartości domyślnych 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, a następnie zatrzymaj przetwarzanie.
3. Utwórz pętlę w programie, wywołując wywołania MQPUT, aż do momentu umieszczenia w kolejce wymaganej liczby komunikatów. Jeśli wywołanie MQPUT nie powiedzie się, pętla jest porzucona na początku, nie są wykonywane żadne dalsze wywołania MQPUT, a zwracane są kody zakończenia i przyczyny.
4. Zamknij kolejkę, korzystając z wywołania MQCLOSE z uchwytem obiektu zwróconego w kroku “2” na stronie 1292. Jeśli wywołanie nie powiedzie się, wydrukuj kody zakończenia i przyczyny.
5. Odłącz się od menedżera kolejek przy użyciu wywołania MQDISC z uchwytem połączenia zwróconego w kroku “1” na stronie 1292. Jeśli wywołanie nie powiedzie się, wydrukuj kody zakończenia i przyczyny.

Uwaga: Jeśli próbka jest uruchamiana w środowisku produktu CICS , nie ma potrzeby wydawania wywołania MQDISC.

z/OS *Przykłady umieszczania dla środowiska wsadowego w systemie z/OS*

W tym temacie opisano sposób uwzględniania przykładów umieszczania dla środowiska wsadowego.

Aby uruchomić przykłady, edytuj i uruchom przykładowy kod JCL zgodnie z opisem w sekcji "Przygotowywanie i uruchamianie przykładowych aplikacji dla środowiska wsadowego w systemie z/OS" na stronie 1280.

Programy przyjmują następujące parametry w programie EXEC PARM, rozdzielając je 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 dopełniania do zapisu w komunikacie (1 znak)
5. Liczba znaków do zapisania w komunikacie (maksymalnie 4 cyfry)
6. Trwałość komunikatu (1 znak: P dla trwałego lub N dla nietrwały)

Jeśli wszystkie te parametry zostaną wprowadzone błędnie, zostaną wyświetlone odpowiednie komunikaty o błędach.

Wszystkie komunikaty z przykładów są zapisywane w zestawie danych SYSPRINT.

Użycie notatek

- Aby zachować proste przykłady, występują niewielkie różnice funkcjonalne między wersjami językową. Różnice te są jednak zminimalizowane, jeśli używany jest układ parametrów wyświetlanych w przykładowych uruchomieniowych JCL, CSQ4BCJRi CSQ4BVJR. Żadne z różnic nie odnosi się do interfejsu MQI.
- CSQ4BCK1 pozwala na wprowadzenie więcej niż czterech cyfr dla liczby wysłanych komunikatów i długości komunikatów.
- W przypadku dwóch pól numerycznych wprowadź dowolną cyfrę z zakresu od 1 do 9999. Wprowadzona wartość powinna być liczbą dodatnią. Na przykład, aby umieścić pojedynczy komunikat, jako wartość można wprowadzić wartość 1, 01, 001 lub 0001. W przypadku wprowadzenia wartości nieliczbowych lub ujemnych może zostać wyświetlony błąd. Na przykład, jeśli zostanie wprowadzona wartość -1, program w języku COBOL wyśle komunikat 1-bajtowy, ale program C odbierze błąd.
- W przypadku obu programów: CSQ4BCK1 i CSQ4BVK1, należy wprowadzić wartość P w parametrze trwałości, + + PER + +, jeśli komunikat ma być trwały. Jeśli nie powiedzie się to, komunikat będzie nietrwały.

z/OS *Przykłady umieszczania dla środowiska CICS w systemie z/OS*

Ten temat zawiera informacje na temat przykładów umieszczania przykładów dla środowiska CICS .

Transakcje przyjmują następujące parametry rozdzielone przecinkami:

1. Liczba wiadomości (maksymalnie 4 cyfry)
2. Znak dopełniania do zapisu w komunikacie (1 znak)
3. Liczba znaków do zapisania w komunikacie (maksymalnie 4 cyfry)
4. Trwałość komunikatu (1 znak: P dla trwałego lub N dla nietrwały)
5. Nazwa kolejki docelowej (48 znaków)

Jeśli wszystkie te parametry zostaną wprowadzone błędnie, zostaną wyświetlone odpowiednie komunikaty o błędach.

W przypadku przykładu COBOL wywołaj przykład Put w środowisku CICS , wpisując:

```
MVPT,9999,*,9999,P,QUEUE.NAME
```

W przypadku przykładu C wywołaj przykład Put w środowisku CICS , wpisując:

```
MCPT,9999,*,9999,P,QUEUE.NAME
```

Wszystkie komunikaty z przykładów są wyświetlane na ekranie.

Użycie notatek

- Aby zachować proste przykłady, występują niewielkie różnice funkcjonalne między wersjami językową. Żadne z różnic nie odnosi się do 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 należy nacisnąć klawisz CLEAR.
- W przypadku dwóch pól numerycznych wprowadź 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. W przypadku wprowadzenia wartości nieliczbowych lub ujemnych może zostać wyświetlony błąd. Na przykład, jeśli zostanie wprowadzona wartość -1, program w języku COBOL wyśle komunikat 1-bajtowy, a program w języku C zakończy działanie z błędem z funkcji malloc ().
- W przypadku obu programów, CSQ4CCK1 i CSQ4CVK1, wpisz P w parametrze trwałości, jeśli komunikat ma być trwały. W przypadku komunikatów nietrwałych należy wprowadzić wartość N w parametrze trwałości. Jeśli zostanie wprowadzona 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 *Pobieranie przykładów w systemie z/OS*

Program pobierania próbek pobiera komunikaty z kolejki za pomocą wywołania MQGET.

Programy źródłowe są dostarczane w językach C i COBOL w środowiskach wsadowych i CICS (patrz Tabela 179 na stronie 1281 i Tabela 186 na stronie 1286).

z/OS *Projekt przykładu Pobierz w systemie z/OS*

Ta sekcja zawiera informacje na temat projektowania przykładu pobierania oraz uwag dotyczących użycia do rozważenia.

Przepływ przez logikę programu jest następujący:

1. Połącz się z menedżerem kolejek przy użyciu wywołania MQCONN. Jeśli wywołanie nie powiedzie się, wydrukuj kody zakończenia i przyczyny, a następnie zatrzymaj przetwarzanie.

   Uwaga: Jeśli próbka jest uruchamiana w środowisku produktu CICS , nie ma potrzeby wydawania wywołania MQCONN. Jeśli zostanie to wykonalne, zwraca wartość DEF_HCONN. Za pomocą uchwytu połączenia MQHC_DEF_HCONN można korzystać w przypadku wywołań MQI.
2. Otwórz kolejkę za pomocą wywołania MQOPEN z opcjami MQOO_INPUT_SHARED i MQOO_BROWSE. Po wejściu do tego wywołania program korzysta z uchwytu połączenia, który jest zwracany w kroku “1” na stronie 1292. W przypadku struktury deskryptora obiektu (MQOD) używa ona wartości domyślnych 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, a następnie zatrzymaj przetwarzanie.
3. Utwórz pętlę w programie, wywołując wywołania MQGET, aż do momentu pobrania wymaganej liczby komunikatów z kolejki. Jeśli wywołanie MQGET nie powiedzie się, pętla jest porzucona na początku,

nie są wykonywane żadne kolejne wywołania MQGET, a zwracane są 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 znajduje się w sekcji MQGET. Dla każdego komunikatu wyświetlany jest numer komunikatu, po którym następuje długość komunikatu i dane komunikatu.

4. Zamknij kolejkę, korzystając z wywołania MQCLOSE z uchwytem obiektu zwróconego w kroku “2” na stronie 1292. Jeśli wywołanie nie powiedzie się, wydrukuj kody zakończenia i przyczyny.
5. Odłącz się od menedżera kolejek przy użyciu wywołania MQDISC z uchwytem połączenia zwróconego w kroku “1” na stronie 1292. Jeśli wywołanie nie powiedzie się, wydrukuj kody zakończenia i przyczyny.

   Uwaga: Jeśli próbka jest uruchamiana w środowisku produktu CICS , nie ma potrzeby wydawania wywołania MQDISC.

Użycie notatek

- Aby zachować proste przykłady, występują niewielkie różnice funkcjonalne między wersjami językową. Różnice te są jednak zminimalizowane, jeśli używany jest układ parametrów wyświetlanych w przykładowych uruchomieniowych JCL, CSQ4BCJRi CSQ4BVJR. Żadne z różnic nie odnosi się do interfejsu MQI.
- CSQ4BCJ1 umożliwia wprowadzenie więcej niż czterech cyfr dla liczby pobieranych komunikatów.
- Komunikaty dłuższe niż 64 kB są obcinane.
- CSQ4BCJ1 może poprawnie wyświetlać komunikaty znakowe, ponieważ jest wyświetlany tylko do momentu, gdy zostanie wyświetlony pierwszy znak NULL (\0).
- W polu liczbowym numeru komunikatu wprowadź dowolną cyfrę z zakresu od 1 do 9999. Wprowadzona wartość powinna być liczbą dodatnią. Na przykład, aby uzyskać pojedynczy komunikat, jako wartość można wprowadzić 1, 01, 001 lub 0001. W przypadku wprowadzenia wartości nieliczbowych lub ujemnych może zostać wyświetlony błąd. Na przykład, jeśli zostanie wprowadzona wartość -1, program w języku COBOL pobierze jeden komunikat, ale program w języku C nie pobierze żadnych komunikatów.
- W przypadku 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 + +, w przypadku komunikatów, które mają zostać pobrane w punkcie synchronizacji.

z/OS *Pobieranie przykładów dla środowiska wsadowego w systemie z/OS*

Aby uruchomić przykłady, edytuj i uruchom przykładowy kod JCL zgodnie z opisem w sekcji “Przygotowywanie i uruchamianie przykładowych aplikacji dla środowiska wsadowego w systemie z/OS” na stronie 1280.

Programy przyjmują następujące parametry w programie EXEC PARM, rozdzielając je 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 wiadomości (1 znak: B do przeglądania lub D do destrukcyjnego pobierania komunikatów)
5. Element sterujący syncpoint (1 znak: S dla punktu synchronizacji lub N dla braku punktu synchronizacji)

W przypadku nieprawidłowego wprowadzenia któregokolwiek z tych parametrów zostaną wyświetlone odpowiednie komunikaty o błędach.

Dane wyjściowe z przykładów 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 *Pobieranie przykładów dla środowiska CICS w systemie z/OS*

Specjalne uwagi dotyczące pobierania przykładów dla środowiska CICS .

Transakcje przyjmują następujące parametry w EXEC PARM, rozdzielając je przecinkami:

1. Liczba wiadomości do pobrania (maksymalnie cztery cyfry)
2. Opcja przeglądania/pobierania wiadomości (jeden znak: B do przeglądania lub D do destrukcyjnego pobierania komunikatów)
3. Element sterujący syncpoint (jeden znak: S dla punktu synchronizacji lub N dla braku punktu synchronizacji)
4. Nazwa kolejki docelowej (48 znaków)

W przypadku nieprawidłowego wprowadzenia któregokolwiek z tych parametrów zostaną wyświetlone odpowiednie komunikaty o błędach.

W przypadku przykładu COBOL wywołaj próbę pobrania w środowisku CICS , wpisując:

```
MVGT,9999,B,S,QUEUE.NAME
```

W przypadku przykładu C wywołaj próbę pobrania w środowisku CICS , wprowadzając komendę:

```
MCGT,9999,B,S,QUEUE.NAME
```

Gdy komunikaty są pobierane z kolejki, są one umieszczane w tymczasowej kolejce pamięci masowej CICS o takiej samej nazwie, jak transakcja CICS (na przykład MCGT dla próbki C).

Poniżej przedstawiono przykładowe dane wyjściowe pobierania próbek:

```
**************************** TOP OF QUEUE ************************
000000000 : 000000010: **********
000000001 : 000000010 :**********
************************** BOTTOM OF QUEUE **********************
```

Użycie notatek

- Aby zachować proste przykłady, występują niewielkie różnice funkcjonalne między wersjami językową. Żadne z różnic nie odnosi się do 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 należy nacisnąć klawisz CLEAR.

- CSQ4CCJ1 może poprawnie wyświetlać komunikaty znakowe, ponieważ jest wyświetlany tylko do momentu, gdy zostanie wyświetlony pierwszy znak NULL (\0).
- Dla pola liczbowego wprowadź 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. Jeśli zostanie wprowadzona wartość nieliczbowa lub ujemna, może zostać wyświetlony błąd.
- Komunikaty dłuższe niż 24 526 bajtów w języku C i 9 950 bajtów w języku COBOL są obcinane. Jest to spowodowane tym, że używane są tymczasowe kolejki pamięci masowej CICS .
- W przypadku obu programów, CSQ4CCK1 i CSQ4CVK1, wpisz B w parametrze get, jeśli chcesz przeglądać komunikaty, w przeciwnym razie wpisz D. Powoduje to destrukcyjne wywołania MQGET. Jeśli zostanie wprowadzona inna wartość, zostanie wyświetlony komunikat o błędzie.
- W przypadku obu programów: CSQ4CCJ1 i CSQ4CVJ1, wprowadź wartość S w parametrze syncpoint w celu pobrania komunikatów w punkcie synchronizacji. Jeśli w parametrze syncpoint zostanie wprowadzona wartość N, wywołania MQGET są wysyłane z punktu synchronizacji. Jeśli zostanie wprowadzona inna wartość, zostanie wyświetlony komunikat o błędzie.

z/OS *Przykład przeglądania w systemie z/OS*

Przykład przeglądania to aplikacja wsadowa, która demonstruje sposób przeglądania komunikatów w kolejce przy użyciu wywołania MQGET.

Aplikacja przechodzi przez wszystkie komunikaty znajdujące się w kolejce, drukując pierwsze 80 bajtów każdego z nich. Aplikacja ta może być używana do wyszukiwania komunikatów w kolejce bez ich zmieniania.

Programy źródłowe i przykładowe uruchamiane JCL są dostarczane w językach COBOL, assembler, PL/I i C (patrz Tabela 180 na stronie 1281).

Aby uruchomić aplikację, należy zmodyfikować i uruchomić przykładowy skrypt JCL, zgodnie z opisem w sekcji “Przygotowywanie i uruchamianie przykładowych aplikacji dla środowiska wsadowego w systemie z/OS” na stronie 1280. Można sprawdzić komunikaty w jednej z własnych kolejek, określając nazwę kolejki w uruchomionym JCL.

Podczas uruchamiania aplikacji (i niektórych komunikatów w kolejce), 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 nagłówki i tylko komunikat `Koniec raportu` . Jeśli wystąpi błąd z dowolnym z wywołań MQI, kody zakończenia i przyczyny są dodawane do wyjściowego zestawu danych.

z/OS *Projekt przykładu przeglądania w systemie z/OS*

Przykładowa aplikacja Przeglądaj korzysta z pojedynczego modułu programu; jedna z nich jest dostępna w każdym obsługiwanym języku programowania.

Przepływ przez logikę programu jest następujący:

1. Otwórz zestaw danych wydruku i wydrukuj wiersz tytułu raportu. Sprawdź, czy nazwy menedżera kolejek i kolejki zostały przekazane z uruchomionego zadania JCL. Jeśli obie nazwy zostały przekazane, należy wydrukować wiersze raportu, które zawierają nazwy. Jeśli nie, wydrukuj komunikat o błędzie, zamknij zestaw danych wydruku i zatrzymaj przetwarzanie.

   Sposób, w jaki program testuje parametry, które jest przekazywane z JCL, zależy od języka, w którym napisany jest program; więcej informacji na ten temat zawiera sekcja "Uwagi dotyczące projektowania zależne od języka w systemie z/OS" na stronie 1297.
2. Połącz się z menedżerem kolejek przy użyciu wywołania MQCONN. Jeśli to wywołanie nie powiedzie się, wydrukuj kod zakończenia i kody przyczyny, zamknij zestaw danych wydruku i zatrzymaj przetwarzanie.
3. Otwórz kolejkę za pomocą wywołania MQOPEN z opcją MQOO_BROWSE. Po wejściu do tego wywołania program korzysta z uchwytu połączenia zwróconego w kroku "2" na stronie 1296. W przypadku struktury deskryptora obiektu (MQOD) używa ona wartości domyślnych dla wszystkich pól z wyjątkiem nazwy kolejki (która została przekazana w kroku "1" na stronie 1296). Jeśli to wywołanie nie powiedzie się, wydrukuj kod zakończenia i kody przyczyny, zamknij zestaw danych wydruku i zatrzymaj przetwarzanie.
4. Należy przejrzeć pierwszy komunikat w kolejce, używając wywołania MQGET. Na wejściu do tego wywołania program określa:
   - Uchwyty połączenia i kolejki z kroków "2" na stronie 1296 i "3" na stronie 1296
   - Struktura MQMD ze wszystkimi polami ustawionym na ich początkowe wartości
   - Dwie opcje:
     - MQGMO_BROWSE_FIRST
     - Komunikat MQGMO_ACCEPT_TRUNCATED_MSG
   - Bufor o wielkości 80 bajtów do przechowywania danych skopiowanych z komunikatu.

   Opcja MQGMO_ACCEPT_TRUNCATED_MSG umożliwia zakończenie połączenia nawet wtedy, gdy komunikat jest dłuższy niż 80-bajtowy bufor określony w wywołaniu. Jeśli komunikat jest dłuższy niż bufor, komunikat jest obcinany w celu dopasowania go do buforu, a kody zakończenia i przyczyny są ustawione w taki sposób, aby były wyświetlane. Próbka została zaprojektowana w taki sposób, aby komunikaty były obcinane do 80 znaków, aby raport był łatwy do odczytania. Wielkość buforu jest ustawiana przez instrukcję `DEFINE` , dzięki czemu można ją łatwo zmienić, jeśli ma być ona ustawiona.
5. Wykonaj następującą pętlę, dopóki wywołanie MQGET nie powiedzie się:
   a. Wydrukuj wiersz raportu pokazujący:
      - Numer kolejny komunikatu (jest to liczba operacji przeglądania).
      - Prawdziwa długość komunikatu (a nie obcięta długość). Ta wartość jest zwracana w polu `DataLength` wywołania MQGET.
      - Pierwsze 80 bajtów danych komunikatu.

   b. Zresetuj pola `MsqId` i `CorrelId` struktury MQMD na wartości NULL.

   c. Przejrzyj następny komunikat, używając wywołania MQGET z tymi dwoma opcjami:
      - MQGMO_BROWSE_NEXT
      - Komunikat MQGMO_ACCEPT_TRUNCATED_MSG
6. Jeśli wywołanie MQGET nie powiedzie się, sprawdź kod przyczyny, aby sprawdzić, czy wywołanie nie powiodło się, ponieważ kursor przeglądania dotarł do końca kolejki. W takim przypadku należy wydrukować komunikat `End of report` (Koniec raportu) i przejść do kroku "7" na stronie 1296 . W przeciwnym razie należy wydrukować kody zakończenia i przyczyny, zamknąć zestaw danych wydruku i zatrzymać przetwarzanie.
7. Zamknij kolejkę, korzystając z wywołania MQCLOSE z uchwytem obiektu zwróconego w kroku "3" na stronie 1296.
8. Odłącz się od menedżera kolejek przy użyciu wywołania MQDISC z uchwytem połączenia zwróconego w kroku "2" na stronie 1296.

9. Zamknij zestaw danych drukowania i zatrzymaj przetwarzanie.

z/OS *Uwagi dotyczące projektowania zależne od języka w systemie z/OS*

Moduły źródłowe są udostępniane w celu przeglądania przykładu w czterech językach programowania.

Istnieją dwie główne różnice między modułami źródłowymi:

- Podczas testowania parametrów przekazywanych z uruchamianego kodu JCL moduły języka COBOL, PL/I i asembler wyszuka przecinek (,). Jeśli zadanie JCL przekazuje produkt `PARM=(,LOCALQ1)`, aplikacja podejmie próbę otwarcia kolejki LOCALQ1 w domyślnym menedżerze kolejek. Jeśli po przecinku (lub bez przecinka) nie ma nazwy, aplikacja zwraca błąd. W module C nie jest wyszukiwany znak przecinka. Jeśli zadanie JCL przekaże jeden parametr (na przykład `PARM=('LOCALQ1')`), moduł C użyje tego jako nazwy kolejki w domyślnym menedżerze kolejek.
- Aby prosty moduł języka asemblera był prosty, używany jest format daty *yy/ddd* (na przykład 05/116) podczas tworzenia raportu wydruku. Pozostałe moduły korzystają z daty kalendarzowej w formacie *mm/dd/rr* .

z/OS *Przykład komunikatu drukowania w systemie z/OS*

Przykład Message Message to aplikacja wsadowa, która demonstruje sposób usunięcia wszystkich komunikatów z kolejki za pomocą wywołania MQGET.

W przykładzie komunikatu drukowania użyto trzech parametrów:

1. Nazwa menedżera kolejek
2. Nazwa kolejki źródłowej
3. Opcjonalny parametr dla właściwości

W przypadku każdego komunikatu są również drukowane pola deskryptora komunikatu, a następnie dane komunikatu. Program drukuje dane zarówno w postaci szesnastkowej, jak i w postaci znaków drukowanych (jeśli są one drukowane). Jeśli znak nie jest drukowalny, program zastępuje go znakiem kropki (.). Programu można używać podczas diagnozowania problemów związanych z aplikacją, która umieszcza komunikaty w kolejce.

Dopuszczalne wartości parametru właściwości to:

Tabela 192. Dopuszczalne wartości parametru właściwości

Wartość	Zachowanie
0	Domyślne zachowanie, tak jak w przypadku produktu IBM WebSphere MQ 6. Właściwości, które są dostarczane do aplikacji, są zależne od atrybutu kolejki **PropertyControl** , z którego pochodzi komunikat.

1	Uchwyt komunikatu jest tworzony i używany razem z MQGET. Właściwości komunikatu, z wyjątkiem tych, które znajdują się w deskryptorze komunikatu (lub rozszerzeniu), są wyświetlane w podobny sposób do deskryptora komunikatu. Na przykład:
	`****Message properties****`
	`  property name: property value`
	Lub jeśli żadne właściwości nie są dostępne:
	`****Message properties****`
	`  None`
	Wartości liczbowe są wyświetlane przy użyciu `printf`, wartości łańcuchowe są otaczane w pojedynczych cudzysłowach, a łańcuchy bajtowe są otoczone znakiem X i apostrofami, co dla deskryptora komunikatu.

2	Określono parametr MQGMO_NO_PROPERTIES, tak aby zwracane były tylko właściwości deskryptora komunikatu.

Tabela 192. Dopuszczalne wartości parametru właściwości (kontynuacja)	
Wartość	**Zachowanie**
3	Określono wartość MQGMO_PROPERTIES_FORCE_MQRFH2 , tak aby wszystkie właściwości zostały zwrócone w danych komunikatu.
4	Właściwość MQGMO_PROPERTIES_COMPATIBILITY jest określona, tak aby wszystkie właściwości mogły zostać zwrócone w zależności od tego, czy właściwość IBM WebSphere MQ 6 jest włączona, w przeciwnym razie właściwości są usuwane.

Istnieje możliwość zmiany aplikacji w taki sposób, aby była ona przeglądanych komunikatów, a nie usuwana z kolejki. W tym celu należy skompilować z opcją -DBROWSE, aby zdefiniować makro BROWSE, zgodnie ze wskazaniem w “Projekt przykładowego komunikatu drukowania w systemie z/OS” na stronie 1299. Kod wykonywalny jest dostępny dla użytkownika w bibliotece SCSQLOAD. Moduł CSQ4BCG0 jest zbudowany przy użyciu modułu -DBROWSE;, moduł CSQ4BCG1 odczytuje niszczycielski odczyt kolejki.

Aplikacja posiada jeden program źródłowy, który jest napisany w języku C. Dostępny jest również przykładowy kod JCL uruchamiania (patrz Tabela 181 na stronie 1282).

Aby uruchomić aplikację, należy zmodyfikować i uruchomić przykładowy skrypt JCL, zgodnie z opisem w sekcji “Przygotowywanie i uruchamianie przykładowych aplikacji dla środowiska wsadowego w systemie z/OS” na stronie 1280. Podczas uruchamiania aplikacji (i niektórych komunikatów w kolejce), wyjściowy zestaw danych wygląda tak, jak w programie Rysunek 144 na stronie 1299.

```
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 144. Przykład raportu z przykładowej aplikacji Drukowanie komunikatów

z/OS *Projekt przykładowego komunikatu drukowania w systemie z/OS*

Przykładowa aplikacja Drukowanie komunikatów używa 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. Połącz się z menedżerem kolejek przy użyciu wywołania MQCONN. Jeśli to wywołanie nie powiedzie się, wydrukuj kod zakończenia i kody przyczyny, a następnie 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, a nie usuwać je z kolejki, należy skompilować przykład z parametrem -DBROWSE lub dodać `#define BROWSE` na początku źródła. W takim przypadku preprocesor makrodefinicji dodaje linię w programie, która wybiera opcję MQOO_BROWSE w kompilacji.

Po wejściu do tego wywołania program korzysta z uchwytu połączenia zwróconego w kroku “2” na stronie 1299. W przypadku struktury deskryptora obiektu (MQOD) używa ona wartości domyślnych dla wszystkich pól z wyjątkiem nazwy kolejki (która została przekazana w kroku “1” na stronie 1299). Jeśli to wywołanie nie powiedzie się, wydrukuj kod zakończenia i przyczyny, a następnie 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 z kolejnymi wywołaniami MQGET. Jeśli to wywołanie nie powiedzie się, wydrukuj kody zakończenia i przyczyny, a następnie zatrzymaj przetwarzanie.
5. Ustaw opcje pobierania komunikatów, aby odzwierciedlić działanie żądania dla wszystkich właściwości komunikatu.
6. Wykonaj następującą pętlę, dopóki wywołanie MQGET nie powiedzie się:
   a. Zainicjuj bufor do odstępów, aby dane komunikatu nie zostały uszkodzone przez żadne dane znajdujące się już w buforze.
   b. Ustaw wartości pól `MsgId` i `CorrelId` struktury MQMD na wartości NULL, tak aby wywołanie MQGET zaznaczy pierwszy komunikat z kolejki.
   c. Uzyskaj komunikat z kolejki, używając wywołania MQGET. Na wejściu do tego wywołania program określa:
      - Połączenia i uchwyty obiektów są wykonywane w krokach “2” na stronie 1299 i “3” na stronie 1299.
      - Struktura MQMD ze wszystkimi polami ustawionym na ich początkowe wartości. (`MsgId` i `CorrelId` są resetowane do wartości NULL dla każdej wywołania MQGET).
      - Opcja MQGMO_NO_WAIT.

        Uwaga: Jeśli aplikacja ma przeglądać komunikaty, a nie usuwać je z kolejki, należy skompilować przykład z parametrem -DBROWSE lub dodać `#define BROWSE` na początku źródła. W takim przypadku preprocesor makrodefinicji dodaje wiersz w programie, który wybiera opcję MQGMO_BROWSE_NEXT do kompilacji. Jeśli ta opcja jest używana w wywołaniu z kolejką, dla której żaden kursor przeglądania nie był wcześniej używany z bieżącym uchwytem obiektu, kursor przeglądania jest ustawiony logicznie przed pierwszym komunikatem.
      - Bufor o wielkości 64KB do przechowywania danych skopiowanych z komunikatu.
   d. Wywołaj podprocedurę printMD . Spowoduje to wydrukowanie nazwy każdego pola w deskryptorze komunikatu, po którym następuje jego zawartość.
   e. Jeśli w kroku “4” na stronie 1300 utworzono uchwyt komunikatu, wywołaj podprocedurę printProperties w celu wyświetlenia wszystkich właściwości komunikatu.
   f. Należy wydrukować długość komunikatu, a następnie dane komunikatu. Każdy wiersz danych komunikatu ma następujący format:
      - Pozycja względna (szesnastkowo) tej części danych
      - 16 bajtów danych szesnastkowych
      - Ten sam 16 bajtów danych w formacie znakowym, jeśli jest drukowalny (znaki niedrukowalne są zastępowane okresami)
7. Jeśli wywołanie MQGET nie powiedzie się, sprawdź kod przyczyny, aby sprawdzić, czy wywołanie nie powiodło się, ponieważ w kolejce nie ma więcej komunikatów. W takim przypadku należy wydrukować komunikat: Brak kolejnych komunikatów, w przeciwnym razie należy wydrukować kody zakończenia i przyczyny. W obu przypadkach przejdź do kroku “9” na stronie 1301.

   Uwaga: Wywołanie MQGET nie powiedzie się, jeśli znajdzie komunikat, który ma więcej niż 64KB danych. Aby zmienić program w taki sposób, aby obsługiwałby większe komunikaty, można wykonać jedną z następujących czynności:

   - Dodaj opcję MQGMO_ACCEPT_TRUNCATED_MSG do wywołania MQGET, tak aby wywołanie pobiera pierwsze 64KB danych i odrzuci pozostałą część danych.
   - Powoduje, że program pozostawi komunikat w kolejce, gdy znajdzie on jedną z tej ilości danych.

- Zwiększ wielkość buforu

8. Jeśli został utworzony uchwyt komunikatu w kroku “4” na stronie 1300 , wywołaj komendę MQDLTMH, aby ją usunąć.
9. Zamknij kolejkę, korzystając z wywołania MQCLOSE z uchwytem obiektu zwróconego w kroku “3” na stronie 1299.
10. Odłącz się od menedżera kolejek przy użyciu wywołania MQDISC z uchwytem połączenia zwróconego w kroku “2” na stronie 1299.

z/OS *Przykład atrybutów kolejki w systemie z/OS*

Przykład Atrybuty kolejki to aplikacja w trybie konwersacyjnym CICS , która demonstruje użycie wywołań MQINQ i MQSET.

Pokazano w nim sposób zapytania o wartości atrybutów **InhibitPut** i **InhibitGet** kolejek oraz sposób ich zmiany, tak aby programy nie mogły umieszczać komunikatów w kolejce ani pobierania komunikatów z kolejki. W ten sposób można *zablokować* kolejkę w ten sposób, gdy testowany jest program.

Aby zapobiec przypadkowej ingerencji we własne kolejki, ten przykład działa tylko w obiekcie kolejki, który zawiera znaki CSQ4SAMP w pierwszych ośmiu bajtach jego nazwy. Jednak kod źródłowy zawiera komentarze, które pokazują, jak usunąć to ograniczenie.

Programy źródłowe są dostarczane w językach COBOL, assembler i C (patrz Tabela 187 na stronie 1286).

Wersja językowa przykładu asemblera używa kodu, który może być ponownie rozbawiony. Aby to zrobić, 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 można użyć transakcji programu CICS Execution Diagnostic Facility (CEDF) w celu debugowania programu.) Więcej informacji na temat pisania programów reenteralnych zawiera sekcja Kodowanie w języku asemblera System/390.

Aby uruchomić aplikację, uruchom system CICS i użyj następujących transakcji CICS :

- Dla języka COBOL: MVC1
- Dla języka asemblera, MAC1
- Dla C, MCC1

Nazwę dowolnej z tych transakcji można zmienić, zmieniając zestaw danych CSD, o którym mowa w kroku 3.

Wzór próbki

Po uruchomieniu przykładu zostanie wyświetlona mapa ekranu, na której znajdują się pola dla:

- Nazwa kolejki.
- Żądanie użytkownika (poprawne działania to: inquire, allow, lub inhibit)
- Bieżący status operacji put dla kolejki
- Bieżący status operacji pobierania dla kolejki

Pierwsze dwa pola są przeznaczone do wprowadzania danych przez użytkownika. Ostatnie dwa pola są wypełniane przez aplikację: wyświetlane są słowo `INHIBITED` lub słowo `ALLOWED`.

Aplikacja sprawdza poprawność wpisanych wartości w pierwszych dwóch polach. Sprawdza, czy nazwa kolejki rozpoczyna się od znaków CSQ4SAMP i że została wprowadzona jedna z trzech poprawnych żądań w polu Działanie. Aplikacja przekształca wszystkie dane wejściowe na wielkie litery, więc nie można używać żadnych kolejek o nazwach zawierających 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, w zależności od przypadku, do odpowiednich pól ekranu.

Jeśli w polu **Działanie** zostanie wprowadzona wartość `inhibit` , przepływ przez logikę programu będzie następujący:

1. Otwórz kolejkę przy użyciu wywołania MQOPEN z opcją MQOO_SET.
2. Wywołaj komendę MQSET przy użyciu selektorów MQIA_INHIBIT_GET i MQIA_INHIBIT_PUT, a także z wartościami MQQA_GET_INHIBITED i MQQA_PUT_INHIBITED w parametrze **IntAttrs** .
3. Zamknij kolejkę za pomocą wywołania MQCLOSE
4. Przenieś słowo ZABLOKOWANO do odpowiednich pól ekranu

Jeśli w polu **Działanie** zostanie wprowadzona wartość `allow` , wówczas aplikacja będzie wykonała podobne przetwarzanie w tym polu dla żądania zablokowanej. Jedynymi różnicami są ustawienia atrybutów i słów wyświetlanych na ekranie.

Gdy aplikacja otwiera kolejkę, używa domyślnego uchwytu połączenia do menedżera kolejek. (CICS nawiązuje połączenie z menedżerem kolejek podczas uruchamiania systemu CICS). Na tym etapie aplikacja może wychwytywać następujące błędy:

- Aplikacja nie jest połączona z menedżerem kolejek
- Kolejka nie istnieje
- Użytkownik nie ma uprawnień do uzyskiwania dostępu do kolejki
- Aplikacja nie ma uprawnień do otwarcia kolejki.

W przypadku innych błędów MQI w aplikacji wyświetlane są kody zakończenia i przyczyny.

z/OS *Przykład programu Mail Manager w systemie z/OS*

Przykładowa aplikacja programu Mail Manager to pakiet programów demonstrujących wysyłanie i odbieranie komunikatów, zarówno w obrębie jednego środowiska, jak i w różnych środowiskach. Aplikacja jest prostym elektronicznym systemem mailingowym, który umożliwia użytkownikom wymianę komunikatów, nawet jeśli korzystają z różnych menedżerów kolejek.

Aplikacja demonstruje sposób tworzenia kolejek przy użyciu wywołania MQOPEN oraz poprzez umieszczanie 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*

Program Mail Manager jest dostarczany w wersjach działających w dwóch środowiskach. Przygotowanie, które należy wykonać przed uruchomieniem aplikacji, zależy od środowiska, które ma być używane.

Użytkownicy mogą uzyskiwać dostęp do kolejek poczty i kolejek pseudonimów zarówno od TSO, jak i od CICS , tak długo, jak ich identyfikatory użytkowników są takie same w każdym systemie.

Zanim możliwe będzie wysłanie komunikatów do innego menedżera kolejek, należy skonfigurować kanał komunikatów do tego menedżera kolejek. Aby to zrobić, należy użyć funkcji sterowania kanałem IBM MQ, opisanej w sekcji Funkcja sterująca 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 1283.
2. Dostosowuje CLIST do przykładu w celu zdefiniowania:
   - Położenie paneli
   - Położenie pliku komunikatów
   - Położenie modułów ładowania
   - Nazwa menedżera kolejek, który ma być używany z aplikacją

   Dla każdej wersji językowej próbki podano odrębny 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 ISPF. Oznacza to, że nie można użyć przykładowej aplikacji programu Mail Manager po obu stronach podzielonego ekranu. Jeśli tak, to wyniki są nieprzewidywalne.

z/OS *Uruchamianie przykładu programu Mail Manager w systemie z/OS*

Aby uruchomić przykład na serwerze CICS Transaction Server for z/OS , uruchom transakcję MAIL. Jeśli użytkownik nie jest już zalogowany do produktu CICS, aplikacja wyświetli prośbę o wprowadzenie identyfikatora użytkownika, do którego może wysłać wiadomość e-mail.

Po uruchomieniu aplikacji zostanie otwarta kolejka poczty elektronicznej. Jeśli ta kolejka nie istnieje, aplikacja utworzy ją dla użytkownika. Nazwy kolejek poczty elektronicznej mają postać CSQ4SAMP.MAILMGR. *id_użytkownika*, gdzie *id_użytkownika* zależy od środowiska:

W TSO
Identyfikator TSO użytkownika

WCICS
Wpisywanie się do systemu CICS lub identyfikator użytkownika wprowadzony przez użytkownika po uruchomieniu programu Mail Manager.

Wszystkie części nazw kolejek, których używa menedżer poczty, muszą być wielkimi literami.

Następnie aplikacja prezentuje panel menu z opcjami dla:

- Odczytaj przychodzącej wiadomości e-mail
- Wysyłanie wiadomości e-mail
- CREATE NICKNAME

Na panelu menu wyświetlane są również informacje o liczbie wiadomości oczekujących w kolejce poczty elektronicznej. W każdej z opcji menu zostanie wyświetlony kolejny panel:

Odczytaj przychodzącej wiadomości e-mail
Menedżer poczty wyświetla listę komunikatów znajdujących się w kolejce poczty elektronicznej. (Wyświetlane są tylko pierwsze 99 komunikatów w kolejce). Przykład tego panelu znajduje się w sekcji Rysunek 147 na stronie 1307. Po wybraniu komunikatu z tej listy wyświetlana jest treść komunikatu (patrz sekcja Rysunek 148 na stronie 1308).

Wysyłanie wiadomości e-mail
Zostanie wyświetlony panel z zachętą do wprowadzenia:
- Nazwa użytkownika, do którego ma zostać wysłany komunikat.
- Nazwa menedżera kolejek, który jest właścicielem swojej kolejki poczty elektronicznej
- Tekst komunikatu

W polu nazwy użytkownika można wprowadzić identyfikator użytkownika lub pseudonim utworzony za pomocą programu Mail Manager. Pole nazwy menedżera kolejek można pozostawić puste, jeśli kolejka poczty użytkownika należy do tego samego menedżera kolejek, który jest używany, i należy pozostawić to pole puste, jeśli w polu nazwy użytkownika wprowadzono pseudonim:

- Jeśli zostanie podana tylko nazwa użytkownika, program najpierw przyjmuje, że nazwa jest pseudonimem, a następnie wysyła komunikat do obiektu zdefiniowanego przez tę nazwę. Jeśli taki pseudonim nie istnieje, program podejmie próbę wysłania komunikatu do kolejki lokalnej o tej nazwie.
- Jeśli zostanie podana zarówno nazwa użytkownika, jak i nazwa menedżera kolejek, program wyśle komunikat do kolejki poczty zdefiniowanej przez te nazwy.

Na przykład, aby wysłać komunikat do użytkownika JONESM w zdalnym menedżerze kolejek QM12, można wysłać do nich komunikat na dwa sposoby:

- Użyj obu pól, aby określić użytkownika JONESM w menedżerze kolejek QM12.
- Zdefiniuj pseudonim (na przykład MARY) dla tego użytkownika i wyślij im wiadomość, umieszczając MARY w polu nazwy użytkownika i nic w polu nazwy menedżera kolejek.

CREATE NICKNAME

Użytkownik może zdefiniować łatwą do zapamiętania nazwę, której można użyć podczas wysyłania wiadomości do innego użytkownika, z którym często się kontaktujesz. Zostanie wyświetlona prośba o podanie ID użytkownika innego użytkownika oraz nazwy menedżera kolejek, który jest właścicielem kolejki poczty.

Pseudonimy są kolejkami, których nazwy mają postać CSQ4SAMP.MAILMGR. *userid.nickname*, gdzie *ID_użytkownika* jest identyfikatorem użytkownika, a *pseudonim* jest pseudonimem, który ma zostać użyty. W przypadku nazw ustrukturyzowanych w ten sposób użytkownicy mogą mieć własny zestaw pseudonimów.

Typ kolejki tworzony przez program zależy od tego, w jaki sposób wypełniasz pola 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 określona zarówno nazwa użytkownika, jak i nazwa menedżera kolejek (a menedżer kolejek nie jest tym, do którego jest połączony program Menedżer poczty), program tworzy lokalną definicję kolejki zdalnej. Program nie sprawdza istnienia kolejki, do której ta definicja jest tłumaczona, lub nawet, że istnieje zdalny menedżer kolejek.

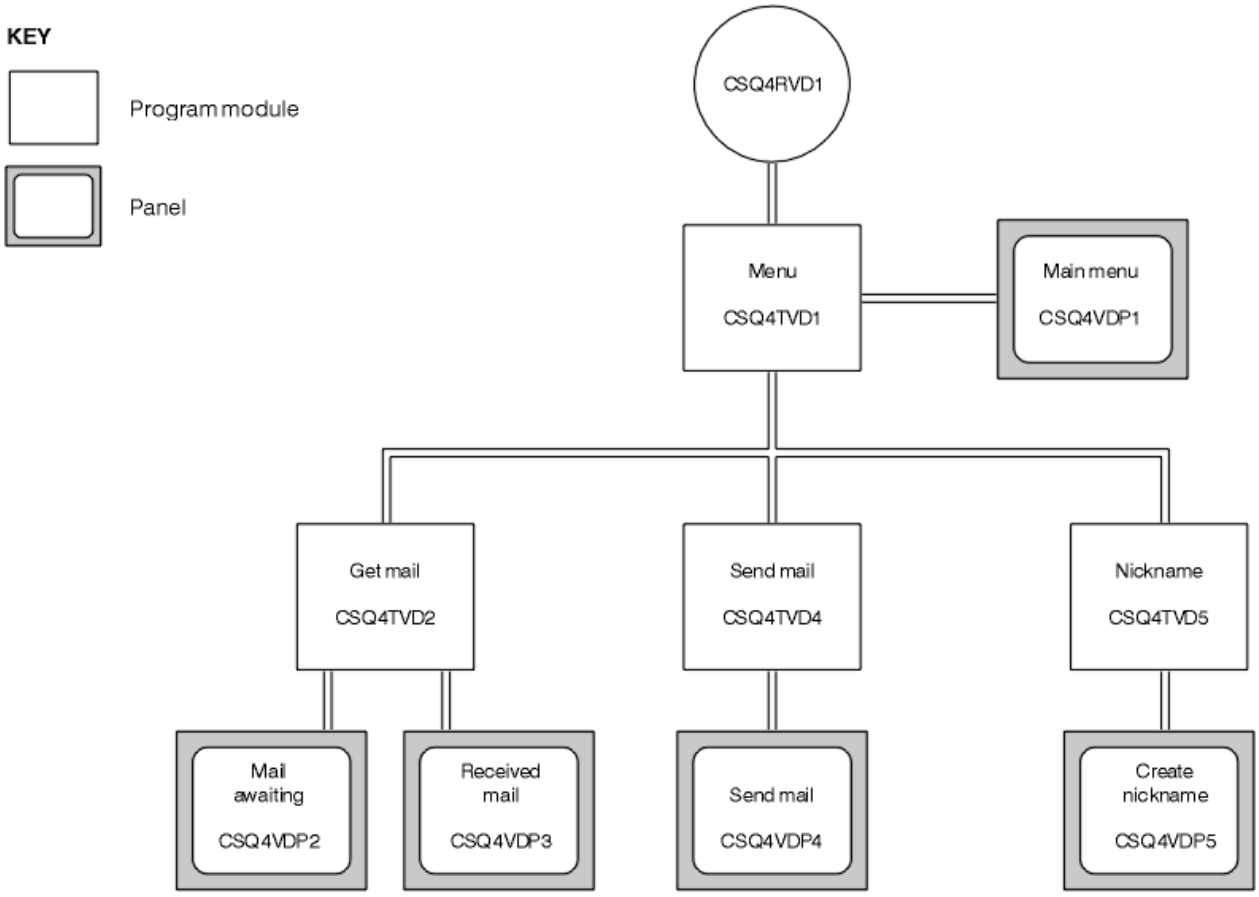
Na przykład, jeśli własny ID użytkownika to SMITHK, a użytkownik tworzy pseudonim MARY dla użytkownika JONESM (który używa zdalnego menedżera kolejek QM12), program pseudonimu tworzy lokalną definicję kolejki zdalnej o nazwie CSQ4SAMP.MAILMGR.SMITHK.MARY. Ta definicja jest tłumaczona na kolejkę poczty Mary, która jest CSQ4SAMP.MAILMGR.JONESM w menedżerze kolejek QM12. Jeśli używany jest sam menedżer kolejek QM12 , program zamiast tego tworzy kolejkę aliasową o tej samej nazwie (CSQ4SAMP.MAILMGR.SMITHK.MARY).

Wersja C aplikacji TSO sprawia, że korzystanie z możliwości obsługi komunikatów ISPF jest większe niż wersja w języku COBOL. Można zauważyć, że różne komunikaty o błędach są wyświetlane w językach C i COBOL.

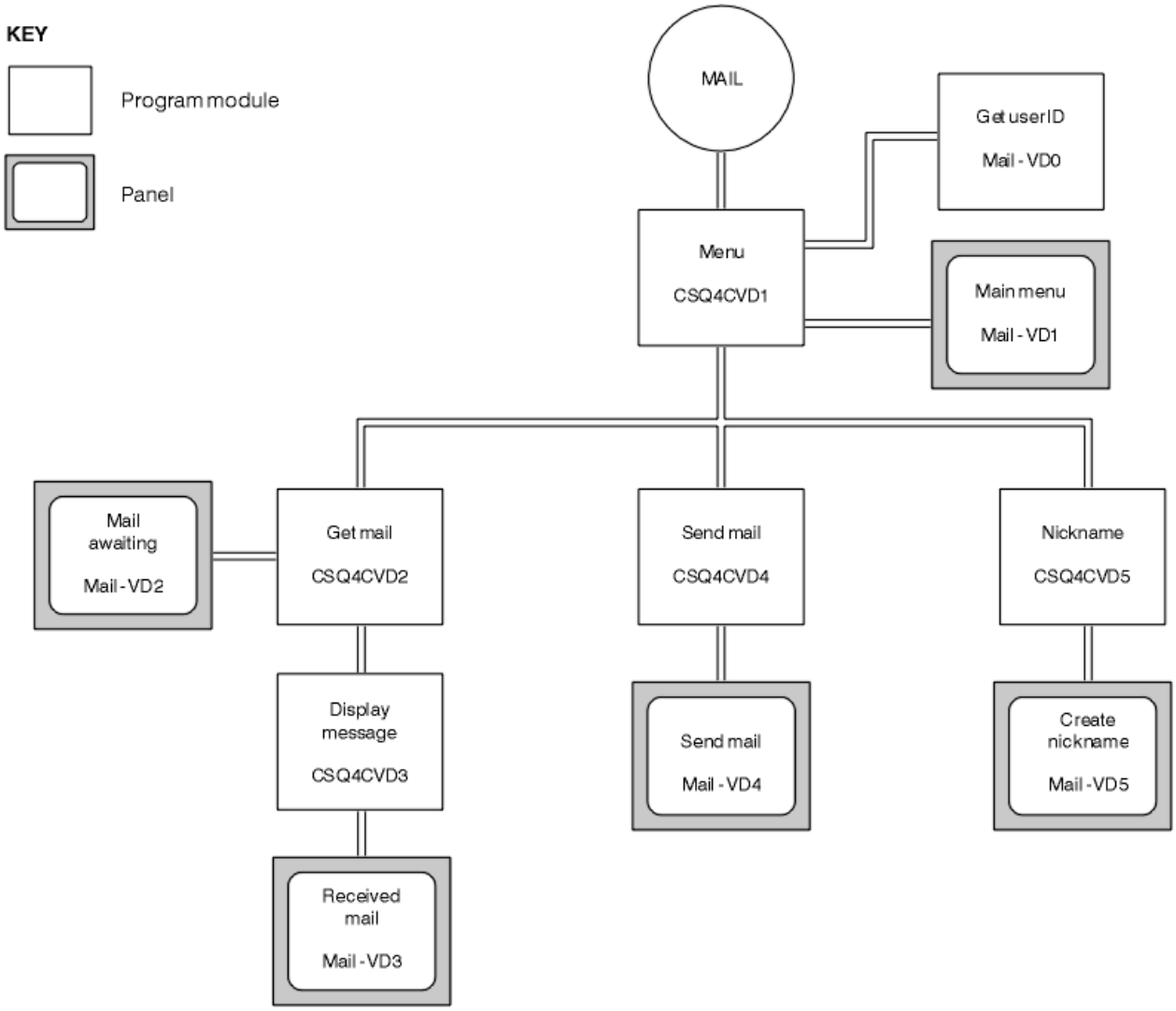
z/OS *Projekt przykładowy programu Mail Manager w systemie z/OS*

W poniższych sekcjach opisano wszystkie programy, które składają się na przykładową aplikację programu Mail Manager.

Relacje między programami i panelami używane przez aplikację są wyświetlane w programie Rysunek 145 na stronie 1305 dla wersji TSO i w produkcie Rysunek 146 na stronie 1306 dla wersji CICS Transaction Server for z/OS .



Rysunek 145. Programy i panele dla wersji TSO programu Mail Manager



Rysunek 146. Programy i panele dla wersji CICS menedżera poczty elektronicznej

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 for CICS) jest programem początkowym w pakiecie. Wyświetla on menu (CSQ4VDP1 dla TSO, VD1 dla CICS) i wywołuje inne programy, gdy są one wybrane z menu.

Program po raz pierwszy uzyskuje identyfikator użytkownika:

- W wersji CICS programu, jeśli użytkownik zalogował się do programu CICS, identyfikator użytkownika jest uzyskiwany za pomocą komendy CICS ASSIGN USERID. Jeśli użytkownik nie zalogował się, program wyświetli panel wpisania się (CSQ4VD0), aby wyświetlić zachętę do wprowadzenia ID użytkownika. W ramach tego programu nie ma żadnych zabezpieczeń; użytkownik może podać dowolny identyfikator 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 przez program ID użytkownika sprawdza, czy użytkownik ma kolejkę poczty (CSQ4SAMP.MAILMGR. *id_użytkownika*). Jeśli kolejka poczty nie istnieje, program tworzy ją przez umieszczenie komunikatu w kolejce wejściowej komend systemowych. Komunikat zawiera komendę

IBM MQ for z/OS DEFINE QLOCAL. Definicja obiektu używana przez tę komendę powoduje ustawienie maksymalnej głębokości kolejki na 9999 komunikatów.

Program tworzy także tymczasową kolejkę dynamiczną, która obsługuje odpowiedzi z kolejki wejściowej komend systemowych. W tym celu program korzysta z wywołania MQOPEN, określając SYSTEM.DEFAULT.MODEL.QUEUE jako szablon kolejki dynamicznej. Menedżer kolejek tworzy tymczasową kolejkę dynamiczną o nazwie, która ma przedrostek CSQ4SAMP; , pozostała część nazwy jest generowana przez menedżer kolejek.

Następnie program otwiera kolejkę poczty użytkownika i znajduje liczbę komunikatów w kolejce, pytając o bieżące zapełnienie kolejki. W tym celu program korzysta z wywołania MQINQ, określając selektor MQIA_CURRENT_Q_DEPTH.

Następnie program wykonuje pętlę, która wyświetla menu i przetwarza wybór dokonany przez użytkownika. Pętla jest zatrzymana, gdy użytkownik naciśnie klawisz PF3 . Po dokonaniu wyboru, uruchamiany jest odpowiedni program; w przeciwnym razie wyświetlany jest komunikat o błędzie.

z/OS *Programy typu "get-mail" 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 funkcje te są wykonywane przez oddzielne programy (CSQ4CVD2 i CSQ4CVD3).

Panel poczty elektronicznej (CSQ4VDP2 dla TSO, VD2 dla CICS ; patrz Rysunek 147 na stronie 1307 dla przykładu) pokazuje wszystkie komunikaty, które znajdują się w kolejce poczty użytkownika. Aby utworzyć tę listę, program korzysta z wywołania MQGET w celu przeglądania wszystkich komunikatów w kolejce, zapisując informacje o każdym z nich. Oprócz wyświetlanych informacji program rejestruje `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 147. Przykład panelu zawierającego listę oczekujących komunikatów

Na panelu Oczekiwanie na wiadomość e-mail użytkownik może wybrać jeden komunikat i wyświetlić jego zawartość (na przykład w sekcji Rysunek 148 na stronie 1308). Program korzysta z wywołania MQGET w celu usunięcia tego komunikatu z kolejki przy użyciu `MsgId` i `CorrelId` , który został zaznaczony przez program podczas przeglądania wszystkich komunikatów. To wywołanie MQGET jest wykonywane przy użyciu opcji MQGMO_SYNCPOINT. Program wyświetli treść komunikatu, a następnie deklaruje punkt synchronizacji: ten zatwierdza wywołanie MQGET, a więc komunikat już nie istnieje.

```
--------------------- IBM MQ for z/OS Sample Programs ---------------------
COMMAND ==>
USERID - NTSFV02
Mail Manager System      QMGR  - VC4
Received Mail

Mail sent from JOHNJ  at VC4

Sent on the 01/06/1993 at 12:52:02
----------------------------------- Message ------------------------------
| HELLO FROM JOHNJ                          |
|                                     |
|                                     |
|                                     |
|                                     |
|                                     |
|                                     |
|                                     |
|                                     |
'-------------------------------------------------------------------------'
```

Rysunek 148. Przykładowy panel pokazujący treść komunikatu

Oczywistym rozszerzeniem funkcji udostępnianej przez menedżera poczty jest nadanie użytkownikowi opcji pozostawienie wiadomości w kolejce po wyświetleniu jej zawartości. Aby to zrobić, należy wycofać wywołanie MQGET, które usuwa komunikat z kolejki po wyświetleniu komunikatu.

z/OS *Program wysyłający pocztę elektroniczną w systemie z/OS*

Po zakończeniu przez użytkownika panelu Wyślij wiadomość e-mail (CSQ4VDP4 dla TSO, VD4 dla CICS), program wysyłający pocztę elektroniczną (CSQ4TVD4 dla TSO, CSQ4CVD4 dla CICS) umieszcza komunikat w kolejce poczty odbiorcy.

Aby to zrobić, program korzysta z wywołania MQPUT1 . Miejsce docelowe komunikatu zależy od tego, w jaki sposób użytkownik wypełnił pola w panelu Wysyłanie wiadomości e-mail:

- Jeśli użytkownik określił tylko nazwę użytkownika, program najpierw przyjmuje, że nazwa jest pseudonimem, a następnie wysyła komunikat do obiektu zdefiniowanego przez tę nazwę. Jeśli taki pseudonim nie istnieje, program podejmie próbę wysłania komunikatu 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 początkowe odstępy z każdego wiersza tekstu komunikatu.

Jeśli wywołanie MQPUT1 zakończy się pomyślnie, program wyświetli komunikat, który pokazuje nazwę użytkownika i nazwę menedżera kolejek, do którego został umieszczony komunikat. Jeśli wywołanie nie powiedzie się, program sprawdza, czy kody przyczyny wskazują, że kolejka lub menedżer kolejek nie istnieją. Są to wartości MQRC_UNKNOWN_OBJECT_NAME i MQRC_UNKNOWN_OBJECT_Q_MGR. Program wyświetli 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 pseudonimu w systemie z/OS*

Gdy użytkownik zdefiniuje pseudonim, program pseudonimu (CSQ4TVD5 dla TSO, CSQ4CVD5 dla CICS) Tworzy kolejkę o pseudonimie jako część jego nazwy.

Program wykonuje tę komendę, umieszczając komunikat w kolejce wejściowej komend systemowych. Komunikat zawiera komendę IBM MQ for z/OS DEFINE QALIAS lub DEFINE QREMOTE. Typ kolejki tworzony przez program zależy od tego, w jaki sposób użytkownik wypełnił pola w 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 jest połączony menedżer poczty, program tworzy kolejkę aliasową.
- Jeśli użytkownik określił zarówno nazwę użytkownika, jak i nazwę menedżera kolejek (a menedżer kolejek nie jest tym, do którego jest połączony menedżer poczty), program tworzy lokalną definicję kolejki zdalnej. Program nie sprawdza istnienia kolejki, do której ta definicja jest tłumaczona, lub nawet, że istnieje zdalny menedżer kolejek.

Program tworzy także tymczasową kolejkę dynamiczną, która obsługuje odpowiedzi z kolejki wejściowej komend systemowych.

Jeśli menedżer kolejek nie może utworzyć kolejki pseudonimu z powodu oczekiwanym przez program (na przykład, kolejka już istnieje), program wyświetli własny komunikat o błędzie. Jeśli menedżer kolejek nie może utworzyć kolejki z powodu, którego program nie oczekuje, program wyświetli do dwóch komunikatów o błędach zwróconych do programu przez serwer komend.

Uwaga: Dla każdego pseudonimu program pseudonimu tworzy tylko kolejkę aliasową lub lokalną definicję kolejki zdalnej. Kolejki lokalne, do których te nazwy kolejek są rozstrzygane, są tworzone tylko wtedy, gdy identyfikator użytkownika zawarty w pseudonimie jest używany do uruchamiania aplikacji Menedżer poczty.

z/OS *Przykład kontroli kredytowej w systemie z/OS*

Przykładowa aplikacja Check Check to pakiet programów demonstrujący sposób korzystania z wielu funkcji udostępnianych przez produkt IBM MQ for z/OS. Pokazuje on, w jaki sposób wiele komponentów aplikacji może przekazywać komunikaty do siebie nawzajem przy użyciu technik kolejkowania komunikatów.

Przykład może być uruchamiany jako autonomiczna aplikacja produktu CICS . Jednak w celu zademonstrowania sposobu zaprojektowania aplikacji kolejkowania komunikatów, która korzysta z urządzeń udostępnianych zarówno przez środowiska CICS , jak i IMS , jeden moduł jest również dostarczany jako program przetwarzania komunikatów wsadowych IMS . To rozszerzenie do przykładu jest opisane w sekcji “Rozszerzenie IMS na przykład kontroli kredytowej w systemie z/OS” na stronie 1320.

Można również uruchomić przykład na więcej niż jednym menedżerze kolejek i wysłać komunikaty między każdą instancją aplikacji. W tym celu należy zapoznać się z “Przykład kontroli kredytowej z wieloma menedżerami kolejek w systemie z/OS” na stronie 1320.

Programy CICS są dostarczane w językach C i COBOL. Pojedynczy program IMS jest dostarczany tylko w języku C. Dostarczone zestawy danych są wyświetlane w serwerach Tabela 189 na stronie 1287 i Tabela 191 na stronie 1289.

Aplikacja demonstruje metodę oceny ryzyka, gdy klienci banku pytają o pożyczki. Aplikacja pokazuje, w jaki sposób bank może pracować na dwa sposoby na przetwarzanie wniosków o kredyt:

- Przy kontaktach bezpośrednio z klientem, pracownicy banku chcą natychmiastowy dostęp do informacji o koncie i kredycie.
- Zajmując się pisemnymi wnioskami, pracownicy banku mogą zgłaszać szereg wniosków o udzielenie informacji o koncie i o ryzyku kredytowym, a także zajmować się odpowiedziami w późniejszym terminie.

Szczegóły dotyczące finansów i zabezpieczeń w aplikacji zostały zachowane w prosty sposób, dzięki czemu techniki kolejkowania komunikatów są jasne.

z/OS *Przygotowywanie i uruchamianie przykładu kontroli kredytowej w systemie z/OS*

Aby przygotować i uruchomić przykład kontroli kredytowej, wykonaj następujące kroki:

1. Utwórz zestaw danych VSAM, który zawiera informacje na temat niektórych przykładowych kont. W tym celu należy zmodyfikować i uruchomić JCL dostarczone w zestawie danych CSQ4FILE.
2. Wykonaj kroki opisane w temacie “Przygotowywanie przykładowych aplikacji dla środowiska CICS w systemie z/OS” na stronie 1285. (Dodatkowe kroki, które należy wykonać, aby użyć rozszerzenia IMS

do przykładu, są opisane w sekcji “Rozszerzenie IMS na przykład kontroli kredytowej w systemie z/OS” na stronie 1320).

3. Uruchom monitor wyzwalacza CKTI (dostarczany z produktem IBM MQ for z/OS) Dla kolejki CSQ4SAMP.INITIATION.QUEUE, korzystając z transakcji CKQC transakcji CICS .
4. Aby uruchomić aplikację, uruchom system CICS i użyj transakcji MVB1.
5. Wybierz opcję **Natychmiast** lub **Zadanie wsadowe** z pierwszego panelu.

   Panele z zapytaniem natychmiastowym i wsadowym są podobne; Rysunek 149 na stronie 1310 wyświetla panel Natychmiastowe Zapytanie.

```
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 149. Natychmiastowy panel uzyskiwania informacji dla przykładowej aplikacji Sprawdzenie uznania

6. Wprowadź numer konta i kwotę pożyczki w odpowiednich polach. Informacje na temat wprowadzania danych w tych polach można znaleźć w sekcji “Wprowadzanie informacji w panelach uzyskiwania informacji” na stronie 1310 .

Wprowadzanie informacji w panelach uzyskiwania informacji

Przykładowa aplikacja Sprawdzenie uznania sprawdza, czy dane wprowadzone w polu **Żądana kwota** w panelach uzyskiwania informacji są w postaci liczb całkowitych.

Jeśli zostanie wprowadzony jeden z następujących numerów kont, aplikacja znajdzie odpowiednią nazwę konta, średnią saldo konta i indeks wiarygodności kredytowej w zestawie danych VSAM CSQ4BAQ: .

- 2222222222
- 3111234329
- 3256478962
- 3333333333
- 3501676212
- 3696879656
- 4444444444
- 5555555555
- 6666666666
- 7777777777

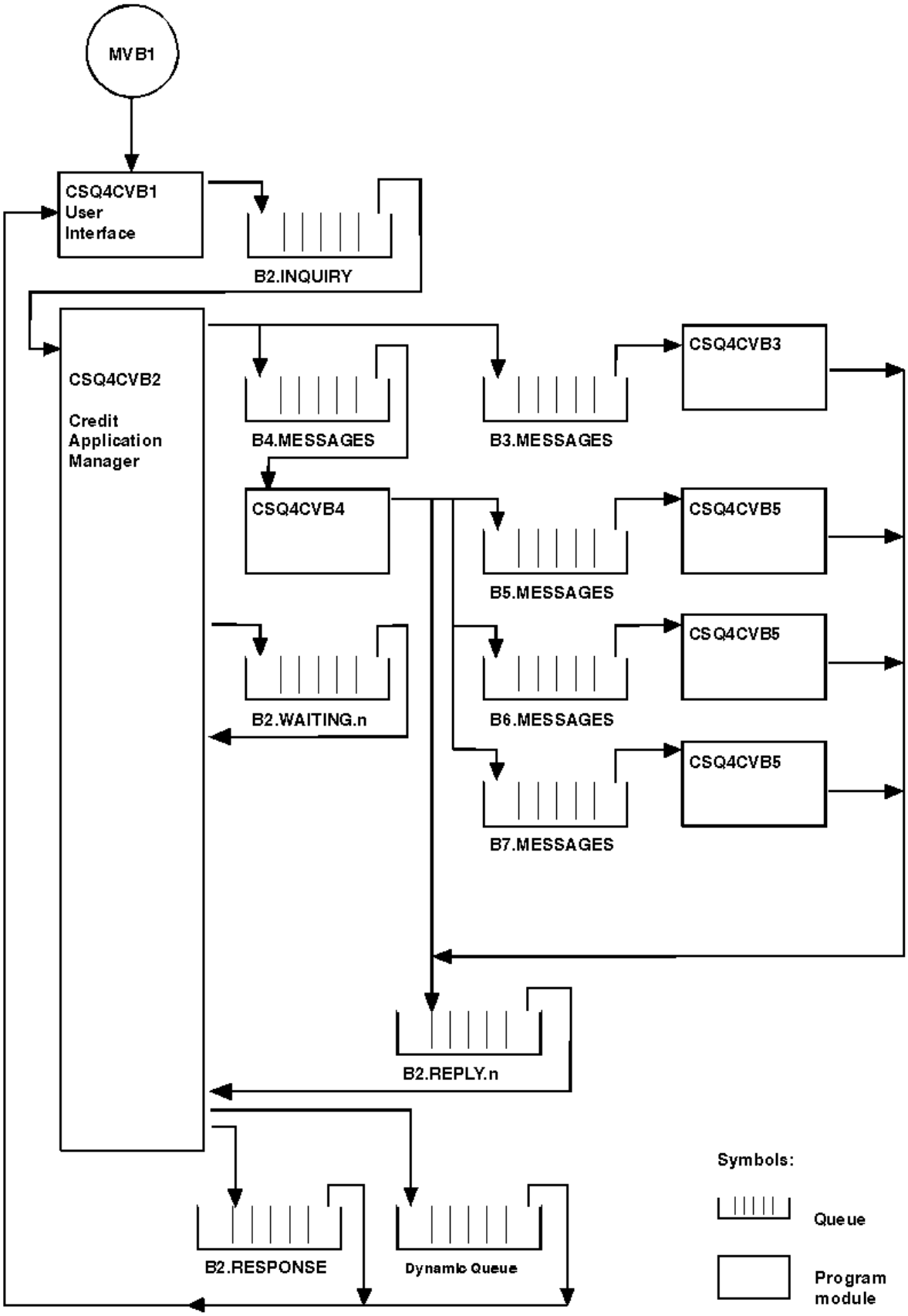
W pozostałych polach można wprowadzić dowolne informacje lub nie. Aplikacja zachowuje wszelkie informacje wprowadzane przez użytkownika i zwracające te same informacje w raportach, które są generowane.

z/OS *Projekt przykładu kontroli kredytowej w systemie z/OS*

W tej sekcji opisano projekt każdego z programów, które składają się na przykładową aplikację kontroli kredytowej.

Więcej informacji na temat niektórych technik, które zostały uwzględnione podczas projektowania aplikacji, zawiera sekcja “Zagadnienia związane z projektowaniem dla przykładu kontroli kredytowej w systemie z/OS” na stronie 1317.

W programie Rysunek 150 na stronie 1312 wyświetlane są programy, które tworzą aplikację, a także kolejki służą do obsługi tych programów. Na tym rysunku przedrostek CSQ4SAMP został pominięty we wszystkich nazwach kolejek, aby ułatwić zrozumienie rysunku.



Rysunek 150. Programy i kolejki dla przykładowej aplikacji Sprawdzenie uznania (tylko programy w języku COBOL)

z/OS *Program interfejsu użytkownika (CSQ4CVB1) w systemie z/OS*

Po uruchomieniu transakcji CICS w trybie konwersacyjnym MVB1, spowoduje to uruchomienie programu interfejsu użytkownika dla aplikacji.

Ten program umieszcza komunikaty z zapytaniem w kolejce CSQ4SAMP.B2.INQUIRY i pobiera odpowiedzi na zapytania z kolejki odpowiedzi, którą określa podczas uzyskiwania zapytania. Z poziomu interfejsu użytkownika można wprowadzać zapytania natychmiastowe lub wsadowe:

- W przypadku natychmiastowych zapytań program tworzy tymczasową kolejkę dynamiczną, której używa jako kolejki odpowiedzi. Oznacza to, że każde zapytanie ma własną kolejkę zwrotną.
- W przypadku zapytań wsadowych program user-interface pobiera odpowiedzi z kolejki CSQ4SAMP.B2.RESPONSE. Dla uproszczenia program otrzymuje odpowiedzi na wszystkie zapytania z tej jednej odpowiedzi do kolejki. Łatwo zauważyć, że bank może chcieć użyć oddzielnej kolejki odpowiedzi dla każdego użytkownika MVB1, tak aby każdy mógł zobaczyć odpowiedzi tylko na te zapytania, które zainicjowały.

Istotne różnice między właściwościami komunikatów używanych w aplikacji w trybie wsadowym i natychmiastowym są następujące:

- W przypadku pracy wsadowej komunikaty mają niski priorytet, dlatego są przetwarzane po wszystkich żądaniach kredytowych wprowadzonych w trybie natychmiastowym. Ponadto komunikaty są trwałe, więc są one odtwarzane, jeśli aplikacja lub menedżer kolejek musi zostać zrestartowany.
- W przypadku natychmiastowej pracy komunikaty mają wysoki priorytet, dlatego są przetwarzane przed wszelkimi żądaniami pożyczki, które są wprowadzane w trybie wsadowym. Ponadto komunikaty nie są trwałe, więc są usuwane, jeśli aplikacja lub menedżer kolejek ma zostać zrestartowany.

Jednak we wszystkich przypadkach właściwości komunikatów żądań dotyczących pożyczek są propagowane w aplikacji. Na przykład wszystkie komunikaty, które wynikają z żądania o wysokim priorytecie, będą miały również wysoki priorytet.

z/OS *Menedżer aplikacji kredytowych (CSQ4CVB2) w systemie z/OS*

Program Menedżer aplikacji kredytowej (CAM) wykonuje większość operacji przetwarzania dla aplikacji Sprawdzenie uznania.

Program CAM jest uruchamiany przez monitor wyzwalacza CKTI (dostarczany z produktem IBM MQ for z/OS), gdy zdarzenie wyzwalające wystąpi w jednej z kolejek CSQ4SAMP.B2.INQUIRY lub kolejka CSQ4SAMP.B2.REPLY. *n*, gdzie *n* jest liczbą całkowitą, która identyfikuje jeden z zestawów kolejek odpowiedzi. Komunikat wyzwalacza zawiera dane, które zawierają nazwę kolejki, w której wystąpiło zdarzenie wyzwalające.

CAM używa kolejek z nazwami w postaci CSQ4SAMP.B2.WAITING.n służy do przechowywania informacji na temat zapytań, które przetwarzali. Nazwy kolejek są tak nazwane, że są połączone z kolejką zwrotną, na przykład kolejkę CSQ4SAMP.B2.WAITING.3 zawiera dane wejściowe dla konkretnego zapytania i kolejki CSQ4SAMP.B2.REPLY.3 zawiera zestaw komunikatów odpowiedzi (z programów, które odpytywać bazy danych) wszystkich związanych z tym samym zapytaniem. Informacje na temat przyczyn tego projektu można znaleźć w sekcji “Oddzielne kolejki zapytań i odpowiedzi w CAM” na stronie 1318.

Logika uruchamiania

Jeśli zdarzenie wyzwalające wystąpi w kolejce CSQ4SAMP.B2.INQUIRY, CAM otwiera kolejkę dla współużytkowanego dostępu. Następnie próbuje otworzyć każdą kolejkę odpowiedzi, dopóki nie zostanie znaleziona wolna. Jeśli nie może znaleźć wolnej kolejki odpowiedzi, CAM rejestruje fakt i przerywa normalne działanie.

Jeśli zdarzenie wyzwalające wystąpi w kolejce CSQ4SAMP.B2.REPLY.n, CAM otwiera kolejkę do wyłącznego dostępu. Jeśli kod powrotu informuje o tym, że obiekt jest już używany, CAM kończy się normalnie. Jeśli wystąpi jakikolwiek inny błąd, CAM zarejestruje błąd i kończy działanie. CAM otwiera odpowiednią kolejkę oczekujących i kolejkę zapytań, a następnie rozpoczyna pobieranie i przetwarzanie komunikatów. Z kolejki oczekujących CAM odzyskuje szczegóły częściowo zakończonych zapytań.

Ze względu na prostotę w tym przykładzie, w programie przechowywane są nazwy używanych kolejek. W środowisku biznesowym nazwy kolejek prawdopodobnie będą przechowywane w pliku, do którego dostęp jest uzyskiwany przez program.

Pobieranie komunikatu z kolejki zapytań

W pierwszej kolejności CAM próbuje pobrać komunikat z kolejki uzyskiwania informacji przy użyciu wywołania MQGET z opcją MQGMO_SET_SIGNAL. Jeśli komunikat jest dostępny natychmiast, komunikat jest przetwarzany; jeśli żaden komunikat nie jest dostępny, ustawiany jest sygnał.

Następnie CAM podejmuje próbę pobrania komunikatu z kolejki odpowiedzi, ponownie używając wywołania MQGET z tą samą opcją. Jeśli komunikat jest dostępny natychmiast, komunikat jest przetwarzany; w przeciwnym razie ustawiany jest sygnał.

Gdy oba sygnały są ustawione, program czeka, aż jeden z sygnałów zostanie opublikowany. Jeśli wysyłany jest sygnał w celu wskazania, że komunikat jest dostępny, to komunikat jest pobierany i przetwarzany. Jeśli sygnał utraci ważność lub menedżer kolejek kończy działanie, program kończy działanie.

Przetwarzanie komunikatu pobranego przez CAM

Komunikat pobrany przez CAM może być jednym z czterech typów:

- Komunikat z zapytaniem
- Komunikat odpowiedzi
- Komunikat propagacji
- Nieoczekiwany lub niepożądany komunikat

CAM przetwarza te komunikaty zgodnie z opisem w sekcji “Przetwarzanie komunikatu pobranego przez CAM w systemie z/OS” na stronie 1315.

Wysyłanie odpowiedzi

Gdy CAM odebrało wszystkie odpowiedzi, które oczekuje na zapytanie, przetwarza odpowiedzi i tworzy pojedynczy komunikat odpowiedzi. Konsoliduje on w jednym komunikacie wszystkie dane ze wszystkich komunikatów odpowiedzi, które mają ten sam produkt `CorrelId`. Ta odpowiedź jest umieszczana w kolejce odpowiedzi określonej w pierwotnym żądaniu pożyczki. Komunikat odpowiedzi jest umieszczany w tej samej jednostce pracy, która zawiera pobieranie końcowego komunikatu odpowiedzi. W tym celu należy uprościć odtwarzanie, upewniając się, że w kolejce CSQ4SAMP.B2.WAITING.n.

Odzyskiwanie częściowo zakończonych zapytań

CAM kopiuje do kolejki CSQ4SAMP.B2.WAITING.n wszystkie komunikaty, które otrzymuje. Ustawia on pola deskryptora komunikatu w następujący sposób:

- *`Priority`* jest określany na podstawie typu komunikatu:
  - Dla komunikatów żądań priorytet = 3
  - Dla datagramów, priorytet = 2
  - Dla komunikatów odpowiedzi priorytet = 1
- *`CorrelId`* jest ustawiony na *`MsgId`* komunikatu żądania pożyczki.
- Inne pola MQMD są kopiowane z tych z odebranego komunikatu

Po zakończeniu uzyskiwania informacji komunikaty dotyczące konkretnego zapytania są usuwane z kolejki oczekujących podczas przetwarzania odpowiedzi. Oznacza to, że w dowolnym momencie kolejka oczekujących zawiera wszystkie komunikaty istotne dla zapytań w toku. Komunikaty te są używane do odzyskiwania szczegółów dotyczących zapytań w toku, jeśli program musi zostać zrestartowany. Różne priorytety są ustawione w taki sposób, że komunikaty z zapytaniem są odtwarzane przed propagacją lub odpowiedziami na odpowiedź.

z/OS *Przetwarzanie komunikatu pobranego przez CAM w systemie z/OS*

Komunikat pobrany przez menedżera aplikacji kredytowych (Credit Application Manager-CAM) może być jednym z czterech typów. Sposób, w jaki proces CAM przetwarza komunikat, zależy od jego typu.

Komunikat pobrany przez CAM może być jednym 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. Tworzy on komunikat z zapytaniem dla każdego wniosku o pożyczkę.

W przypadku wszystkich wniosków o pożyczkę, CAM żąda średniego salda rachunku sprawdzanego przez klienta. W tym celu należy umieścić komunikat żądania w kolejce aliasowej CSQ4SAMP.B2.OUTPUT.ALIAS. Ta nazwa kolejki jest tłumaczona na kolejkę CSQ4SAMP.B3.MESSAGES, które jest przetwarzane przez program do sprawdzania kont, CSQ4CVB3. Gdy CAM umieszcza komunikat w tej kolejce aliasowej, określa ona odpowiednią wartość CSQ4SAMP.B2.REPLY.n kolejka dla kolejki odpowiedzi. W tym miejscu używana jest kolejka aliasowa, dzięki czemu program CSQ4CVB3 może być łatwo zastąpiony przez inny program, który przetwarza kolejkę podstawową o innej nazwie. W tym celu należy ponownie zdefiniować kolejkę aliasową w taki sposób, aby jej nazwa została rozstrzygana do nowej kolejki. Można również przypisać różne uprawnienia dostępu do kolejki aliasowej i do kolejki podstawowej.

Jeśli użytkownik zażąda kredytu, który jest większy niż 10000 jednostek, CAM inicjuje również sprawdzanie innych baz danych. W tym celu należy umieścić komunikat żądania w kolejce CSQ4SAMP.B4.MESSAGES, który jest przetwarzany przez program dystrybucji, 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 put (put). Dodatkowo do kolejki odpowiedzi wysyłany jest komunikat propagacji, który umożliwia określenie liczby wysłanych komunikatów propagacji.

W środowisku biznesowym program dystrybucji prawdopodobnie przeformatowałby dane dostarczone w celu dopasowania do formatu wymaganego przez każdy inny typ konta bankowego.

Dowolna z kolejek, do których odwołuje się system zdalny, może znajdować się w systemie zdalnym.

Dla każdego komunikatu z zapytaniem CAM inicjuje pozycję w tabeli rekordów zapytań rezydentnych (IRT) rezydentnej. Ten rekord zawiera:

- `MsgId` komunikatu z zapytaniem
- W polu ReplyExp (ReplyExp) oczekiwana liczba odpowiedzi (równa liczbie wysłanych komunikatów)
- W polu ReplyRec liczba odebranych odpowiedzi (zero na tym etapie)
- W polu PropsOut (PropsOut) wskazanie, czy oczekiwany jest komunikat propagacji.

CAM kopiuje komunikat z zapytaniem do oczekującej kolejki z:

- `Priority` ustawione na 3
- `CorrelId` ustawione na `MsgId` komunikatu z zapytaniem
- Pozostałe pola deskryptora komunikatu ustawione na wartość dla komunikatu z zapytaniem

Komunikat propagacji

Komunikat propagacji zawiera liczbę kolejek, do których program dystrybucji przekazał zapytanie. Komunikat jest przetwarzany w następujący sposób:

1. Dodaj do pola ReplyExp odpowiedniego rekordu w obszarze IRT, ile komunikatów jest wysłanych. Te informacje znajdują się w komunikacie.
2. Zwiększ o 1 pole ReplyRec rekordu w IRT.
3. Zmniejsz o 1 pole PropsOut rekordu w ramach IRT.
4. Skopiuj wiadomość do oczekującej kolejki. CAM ustawia wartość `Priority` na 2 oraz pozostałe pola deskryptora komunikatu na te, które są wyświetlane w komunikacie propagacji.

Komunikat odpowiedzi

Komunikat odpowiedzi zawiera odpowiedź na jeden z żądań do programu do sprawdzania kont lub do jednego z programów obsługi zapytania agencyjnego. Komunikaty odpowiedzi są przetwarzane w następujący sposób:

1. Zwiększ o 1 pole ReplyRec rekordu w IRT.
2. Skopiuj komunikat do kolejki oczekującej z `Priority` ustawionym na 1, a pozostałe pola deskryptora komunikatu ustawionym na te z komunikatu odpowiedzi.
3. Jeśli ReplyRec = ReplyExp, a PropsOut = 0, ustaw flagę 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 put nie powiedzie się, komunikat zostanie utracony, a program będzie kontynuowany. Więcej informacji na temat projektu tej części programu można znaleźć w sekcji "Sposób obsługi nieoczekiwanych komunikatów przez przykład" na stronie 1318.

z/OS *Sprawdzanie-program konta (CSQ4CVB3) w systemie z/OS*

Program checking-account jest uruchamiany przez zdarzenie wyzwalające w kolejce CSQ4SAMP.B3.MESSAGES. Po otwarciu kolejki program pobiera komunikat z kolejki za pomocą wywołania MQGET z opcją oczekiwania, a przedział czasu oczekiwania ustawiony jest na 30 sekund.

Program przeszukuje zestaw danych VSAM CSQ4BAQ dla numeru konta w komunikacie żądania pożyczki. Pobiera ona odpowiednią nazwę konta, średnią saldo i indeks wiarygodności kredytowej, lub zwraca uwagę, że numer konta nie znajduje się w zestawie danych.

Następnie program umieszcza komunikat odpowiedzi (za pomocą wywołania MQPUT1) w kolejce zwrotnej o nazwie określonej w komunikacie żądania pożyczki. Dla tego komunikatu odpowiedzi program:

- Kopiuje `CorrelId` komunikatu żądania pożyczki.
- Korzysta z opcji MQPMO_PASS_IDENTITY_CONTEXT

Program będzie kontynuował pobieranie komunikatów z kolejki do momentu utraty ważności przez okres oczekiwania.

z/OS *Program dystrybucji (CSQ4CVB4) w systemie z/OS*

Program dystrybucyjny jest uruchamiany przez zdarzenie wyzwalające w kolejce CSQ4SAMP.B4.MESSAGES.

W celu zasymulowania 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, których nazwy są następujące: CSQ4SAMP.B *n*.KOMUNIKATY, gdzie *n* to 5, 6 lub 7. W aplikacji biznesowej agencje mogą znajdować się w osobnych lokalizacjach, więc kolejki te mogą być kolejkami zdalnymi. Aby zmodyfikować przykładową aplikację w taki sposób, aby była ona wyświetlana, patrz "Przykład kontroli kredytowej z wieloma menedżerami kolejek w systemie z/OS" na stronie 1320.

Program dystrybucji wykonuje następujące kroki:

1. Z listy nazw pobiera nazwy kolejek, które mają być używane przez program. Program wykonuje ten program za pomocą wywołania MQINQ, aby dowiedzieć się o atrybutach obiektu listy nazw.

2. Otwiera te kolejki, a także CSQ4SAMP.B4.MESSAGES.
3. Wykonuje następującą pętlę do momentu, gdy nie ma więcej komunikatów w kolejce CSQ4SAMP.B4.MESSAGES:
   a. Uzyskaj komunikat przy użyciu wywołania MQGET z opcją oczekiwania, a przedział czasu oczekiwania jest ustawiony na 30 sekund.
   b. Umieść komunikat w każdej kolejce wymienionej na liście nazw, podając nazwę odpowiedniego pliku CSQ4SAMP.B2.REPLY.n kolejka dla kolejki odpowiedzi. Program kopiuje *CorrelId* komunikatu żądania pożyczki do tych komunikatów kopii i korzysta z opcji MQPMO_PASS_IDENTITY_CONTEXT w wywołaniu MQPUT.
   c. Wyślij komunikat datagramu do kolejki CSQ4SAMP.B2.REPLY.n , aby wyświetlić liczbę komunikatów, które zostały pomyślnie umieszczone.
   d. Zadeklaruj punkt synchronizacji.

z/OS *Program query-query (CSQ4CVB5/CSQ4CCB5) w systemie z/OS*

Program query-query jest dostarczany zarówno jako program w języku COBOL, jak i program w języku C. Oba programy mają ten sam projekt. Pokazuje to, że programy różnych typów mogą łatwo współistnieć w aplikacji IBM MQ , a moduły programu, które składają się na taką aplikację, mogą być łatwo zastąpione.

Instancja programu jest uruchamiana przez zdarzenie wyzwalające dla dowolnej z tych kolejek:

- Dla programu w języku COBOL (CSQ4CVB5):
  - CSQ4SAMP.B5.MESSAGES
  - CSQ4SAMP.B6.MESSAGES
  - CSQ4SAMP.B7.MESSAGES
- W przypadku programu C (CSQ4CCB5), kolejka CSQ4SAMP.B8.MESSAGES

Uwaga: Jeśli chcesz użyć programu C, musisz zmienić definicję listy nazw CSQ4SAMP.B4.NAMELIST , aby zastąpić kolejkę CSQ4SAMP.B7.MESSAGES z CSQ4SAMP.B8.MESSAGES. Aby to zrobić, można użyć dowolnego z następujących elementów:

- Panele kontrolne i kontrolne serwera IBM MQ for z/OS
- Komenda ALTER NAMELIST
- Program narzędziowy CSQUTIL

Po otwarciu odpowiedniej kolejki program pobiera komunikat z kolejki za pomocą wywołania MQGET z opcją oczekiwania i z odstępem czasu oczekiwania ustawionym na 30 sekund.

Program symuluje wyszukiwanie bazy danych agencji, przeszukując zestaw danych VSAM CSQ4BAQ dla numeru konta, które zostało przekazane w komunikacie żądania pożyczki. Następnie buduje odpowiedź, która zawiera nazwę kolejki, która jest używana, oraz indeks wiarygodności kredytowej. Aby uprościć przetwarzanie, wybierany jest losowo indeks wiarygodności kredytowej.

Podczas umieszczania komunikatu odpowiedzi program korzysta z wywołania MQPUT1 i:

- Kopiuje `CorrelId` komunikatu żądania pożyczki.
- Korzysta z opcji MQPMO_PASS_IDENTITY_CONTEXT

Program wysyła komunikat odpowiedzi do kolejki odpowiedzi o nazwie określonej w komunikacie żądania pożyczki. (Nazwa menedżera kolejek, który jest właścicielem kolejki odpowiedzi, jest również określona w komunikacie żądania pożyczki.)

z/OS *Zagadnienia związane z projektowaniem dla przykładu kontroli kredytowej w systemie z/OS*

Uwagi dotyczące projektu dla przykładu kontroli kredytowej.

Ten temat zawiera informacje na temat:

- "Oddzielne kolejki zapytań i odpowiedzi w CAM" na stronie 1318
- "Sposób obsługi błędów przez przykład" na stronie 1318

Oddzielne kolejki zapytań i odpowiedzi w CAM

Aplikacja może używać pojedynczej kolejki dla zapytań i odpowiedzi, ale została zaprojektowana w taki sposób, aby używała oddzielnych kolejek z następujących powodów:

- W przypadku, gdy program obsługuje maksymalną liczbę zapytań, dalsze zapytania mogą być pozostawiane w kolejce. Jeśli używana jest pojedyncza kolejka, należy ją usunąć z kolejki i zapisać w innym miejscu.
- Inne instancje procesu CAM mogą być uruchamiane automatycznie w celu obsługi tej samej kolejki, jeśli ruch komunikatów był na tyle wysoki, aby można było je uzyskać. Ale program musi śledzić zapytania w toku, i aby to zrobić, musi odzyskać wszystkie odpowiedzi na pytania, które zainicjował. Jeśli używana jest tylko jedna kolejka, program będzie musiał przeglądać komunikaty, aby sprawdzić, czy były one przeznaczone dla tego programu, czy dla innego programu. W ten sposób operacja będzie znacznie mniej wydajna.

  Aplikacja może obsługiwać wiele CAMs i efektywnie odtwarzać zapytania w toku za pomocą sparowanych kolejek zwrotnych i oczekujących.
- Program może czekać na wiele kolejek efektywnie za pomocą sygnalizowania.

Sposób obsługi błędów przez przykład

Program interfejsu użytkownika obsługuje błędy, zgłaszając je bezpośrednio do użytkownika.

Inne programy nie mają interfejsów użytkownika, więc muszą obsługiwać błędy w inny sposób. Ponadto, w wielu sytuacjach (na przykład, jeśli wywołanie MQGET nie powiedzie się) te inne programy nie znają tożsamości użytkownika aplikacji.

Inne programy umieszczają komunikaty o błędach w tymczasowej kolejce pamięci masowej CICS o nazwie CSQ4SAMP. Tę kolejkę można przeglądać za pomocą podanej w produkcie CICStransakcji CEBR. Programy zapisują również komunikaty o błędach w dzienniku CSML programu 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 działa, dopóki nie przetworzyła nieoczekiwanego komunikatu. Prawdopodobnie oznacza to, że aplikacja powiadamia operatora, kończy się samo i zapewnia, że nie zostanie ona automatycznie zrestartowana (może to zrobić przez ustawienie wyzwalania). Ten wybór oznacza, że całe przetwarzanie aplikacji może zostać wstrzymane przez jeden nieoczekiwany komunikat, a interwencja operatora jest wymagana do zrestartowania aplikacji.
- Aplikacja usuwa komunikat z kolejki, która służy, umieszcza komunikat w innym miejscu i kontynuuje przetwarzanie. Najlepsze miejsce umieszczenia tego komunikatu znajduje się w kolejce niedostarczonych komunikatów systemowych.

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 go w kolejce niedostarczonych komunikatów, zostanie utracony nieoczekiwany komunikat.

- Długi nieoczekiwany komunikat jest obcinany, jeśli jest dłuższy niż limit dla komunikatów w kolejce niedostarczonych komunikatów lub dłuższy niż wielkość buforu w programie.

Aby upewnić się, że aplikacja sprawnie obsługuje wszystkie zapytania przy minimalnym wpływie na działania zewnętrzne, przykładowa aplikacja kontroli kredytowej korzysta z drugiej opcji. Aby umożliwić oddzielenie przykładu od innych aplikacji, które korzystają z tego samego menedżera kolejek, w przykładzie kontroli kredytowej nie jest używana systemowa kolejka niedostarczonych komunikatów. Zamiast niej korzysta z własnej kolejki niedostarczonych komunikatów. Ta kolejka ma nazwę CSQ4SAMP.DEAD.QUEUE. Próbka obcina wszystkie komunikaty, które są dłuższe niż obszar buforu udostępniony dla przykładowych programów. Za pomocą aplikacji Przeglądaj przykładową można przeglądać komunikaty w tej kolejce lub użyć przykładowej aplikacji Drukowanie komunikatów w celu wydrukowania komunikatów wraz z ich deskryptorami komunikatów.

Jeśli jednak próbka zostanie rozszerzona o wiele menedżerów kolejek, nieoczekiwane komunikaty lub komunikaty, których nie można dostarczyć, można umieścić w kolejce niedostarczonych komunikatów przez menedżer kolejek.

W jaki sposób przykład używa punktów synchronizacji

Programy w przykładowej aplikacji Check Check deklarują punkty synchronizacji, aby zapewnić, że:

- W odpowiedzi na każdy oczekiwany komunikat wysyłany jest tylko jeden komunikat odpowiedzi.
- Wiele kopii nieoczekiwanych komunikatów nigdy nie jest umieszczanych w kolejce niedostarczonych komunikatów.
- CAM może odzyskać stan wszystkich częściowo zakończonych zapytań, otrzymując trwałe komunikaty z kolejki oczekujących.

Aby to osiągnąć, pojedyncza jednostka pracy służy do pokrywać otrzymany komunikat, przetwarzanie tego komunikatu oraz wszelkie kolejne operacje put.

W jaki sposób przykład używa informacji o kontekście komunikatu

Gdy program interfejsu użytkownika (CSQ4CVB1) wysyła komunikaty, korzysta on z opcji MQPMO_DEFAULT_CONTEXT. Oznacza to, że menedżer kolejek generuje zarówno informacje o kontekście tożsamości, jak i kontekstu pochodzenia. Menedżer kolejek pobiera te informacje z transakcji, która uruchomiła program (MVB1), a także z identyfikatora użytkownika, który uruchomił transakcję.

Gdy CAM wysyła komunikaty z zapytaniem, korzysta z opcji MQPMO_PASS_IDENTITY_CONTEXT. Oznacza to, że informacje dotyczące kontekstu tożsamości umieszczanego komunikatu są kopiowane z kontekstu tożsamości oryginalnego komunikatu z zapytaniem. W przypadku tej opcji informacje o kontekście pochodzenia są generowane przez menedżer kolejek.

Gdy CAM wysyła komunikaty odpowiedzi, korzysta z opcji MQPMO_ALTERNATE_USER_AUTHORITY. Powoduje to, że menedżer kolejek używa alternatywnego ID użytkownika dla sprawdzenia zabezpieczeń, gdy CAM otworzy kolejkę odpowiedzi. CAM korzysta z identyfikatora użytkownika osoby przesyłający oryginalną wiadomość z zapytaniem. Oznacza to, że użytkownicy mogą wyświetlać odpowiedzi tylko na te zapytania, które zostały zainicjowane. Alternatywny identyfikator użytkownika jest uzyskiwane z informacji kontekstu tożsamości w deskryptorze komunikatu oryginalnego komunikatu z zapytaniem.

Gdy programy zapytania (CSQ4CVB3/4/5) wysyłają komunikaty odpowiedzi, korzystają z opcji MQPMO_PASS_IDENTITY_CONTEXT. Oznacza to, że informacje dotyczące kontekstu tożsamości umieszczanego komunikatu są kopiowane z kontekstu tożsamości oryginalnego komunikatu z zapytaniem. W przypadku tej opcji informacje o kontekście pochodzenia 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 nie być takie same, jak te, które są powiązane z przetwarzanego żądania. Aby uzyskać dostęp do tej ewentualnej ekspozycji na zabezpieczenia, programy zapytania mogą używać opcji MQPMO_ALTERNATE_USER_AUTHORITY podczas wprowadzania odpowiedzi. Oznaczałoby to, że każdy użytkownik MVB1 musi mieć uprawnienia do otwierania programu B2.REPLY.n kolejek.

Korzystanie z identyfikatorów komunikatów i korelacji w CAM

Aplikacja musi monitorować postęp przetwarzania wszystkich bieżących zapytań, które są przetwarzane w dowolnym momencie. W tym celu używany jest unikalny identyfikator komunikatu dla każdego komunikatu żądania pożyczki w celu powiązania wszystkich informacji, jakie ma on na temat każdego zapytania.

CAM kopiuje `MsgId` komunikatu z zapytaniem do `CorrelId` wszystkich komunikatów żądań, które wysyła dla tego zapytania. Pozostałe programy znajdujące się w próbie (CSQ4CVB3 -5) kopiują `CorrelId` każdego komunikatu, który otrzymują w `CorrelId` komunikatu odpowiedzi.

z/OS *Przykład kontroli kredytowej z wieloma menedżerami kolejek w systemie z/OS*

Można użyć przykładowej aplikacji Sprawdzenie uznania, aby zademonstrować rozproszoną kolejkowanie, instalując przykład na dwóch menedżerach kolejek i systemach CICS (z każdym menedżerem kolejek połączonym z innym systemem CICS).

Gdy przykładowy program jest zainstalowany, a monitor wyzwalacza (CKTI) działa w każdym systemie, należy:

1. Skonfiguruj łącze komunikacyjne między dwoma menedżerami kolejek. Informacje na temat sposobu wykonania tej czynności zawiera sekcja Konfigurowanie kolejkowania rozproszonego.
2. W przypadku jednego menedżera kolejek utwórz definicję lokalną dla każdej kolejki zdalnej (w innym menedżerze kolejek), która ma być używana. Te kolejki mogą być dowolne z CSQ4SAMP.B *n*.KOMUNIKATY, gdzie *n* to 3, 5, 6 lub 7. (Są to kolejki obsługiwane przez program do sprawdzania kont i program query-query). Więcej informacji na ten temat zawiera sekcja DEFINE QREMOTE (DEFINIOWANIE KOLEJEK QREMOTE) i DEFINE queues (DEFINIOWANIE KOLEJEK).
3. Zmień definicję listy nazw (CSQ4SAMP.B4.NAMELIST), tak aby zawierała nazwy kolejek zdalnych, które mają być używane. Informacje na temat sposobu wykonania tej czynności zawiera sekcja DEFINE NAMELIST.

z/OS *Rozszerzenie IMS na przykład kontroli kredytowej w systemie z/OS*

Wersja programu do sprawdzania konta jest dostarczana jako program do przetwarzania komunikatów wsadowych IMS (BMP). Jest 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. If you replace the CICS version of the checking-account program with the IMS version, you see no difference in the method of using the application.

Aby przygotować i uruchomić wersję produktu IMS , należy wykonać następujące czynności:

1. Wykonaj czynności opisane w sekcji “Przygotowywanie i uruchamianie przykładu kontroli kredytowej w systemie z/OS” na stronie 1309.
2. Wykonaj czynności opisane w sekcji “Przygotowywanie przykładowej aplikacji dla środowiska IMS w systemie z/OS” na stronie 1289.
3. Zmień definicję kolejki aliasowej CSQ4SAMP.B2.OUTPUT.ALIAS , aby rozwiązać problem z kolejką CSQ4SAMP.B3.IMS.MESSAGES (zamiast CSQ4SAMP.B3.MESSAGES). Aby to zrobić, można użyć jednego z następujących elementów:
   - Panele kontrolne i kontrolne serwera IBM MQ for z/OS
   - Komenda ALTER QALIAS .

Innym sposobem korzystania z programu do sprawdzania kont IMS jest użycie jednej z kolejek, które odbierają komunikaty z programu dystrybucyjnego. W dostarczonej formie przykładowej aplikacji Sprawdzenie uznania znajdują się trzy z tych kolejek (B5/6/7.MESSAGES), wszystkie obsługiwane przez program query-query. Ten program wyszukuje zestaw danych VSAM. Aby porównać użycie zestawu danych VSAM i bazy danych IMS , można zamiast tego udostępnić program do obsługi kont IMS w jednej z tych kolejek. Aby to zrobić, należy zmienić definicję listy nazw CSQ4SAMP.B4.NAMELIST , aby zastąpić jedną z opcji CSQ4SAMP.B *n*.Kolejki komunikatów MESSAGES z CSQ4SAMP.B3.IMS.KOMUNIKATY, kolejka. Można użyć jednego z następujących elementów:

- Panele kontrolne i kontrolne serwera IBM MQ for z/OS
- Komenda ALTER NAMELIST .

Następnie można uruchomić przykład z transakcji CICS MVB1. Użytkownik nie widzi różnicy w działaniu lub odpowiedzi. Program IMS BMP zatrzymuje się po odebraniu komunikatu zatrzymania lub po jego dezaktywacji przez pięć minut.

Projekt programu do sprawdzania konta w programie IMS (CSQ4ICB3)

Ten program działa jako BMP. Uruchom program, używając jego JCL przed wysłaniem do niego dowolnych komunikatów produktu IBM MQ .

Program przeszukuje bazę danych IMS pod kątem numeru konta w komunikatach żądania pożyczki. Pobiera ona odpowiednią nazwę konta, saldo średnie i indeks wiarygodności kredytowej.

Program wysyła wyniki wyszukiwania bazy danych do kolejki odpowiedzi określonej w przetwarzanym komunikacie IBM MQ . Zwrócony komunikat dopisuje typ konta i wyniki wyszukiwania do odebranego komunikatu, dzięki czemu transakcja budowania odpowiedzi może potwierdzić, że przetwarzane jest poprawne zapytanie. Komunikat ma postać trzech 79-znakowych grup, w następujący sposób:

```
'Response from CHECKING ACCOUNT for name : JONES J B'
'   Opened 870530, 3-month average balance = 000012.57'
'   Credit worthiness index - BBB'
```

Podczas działania jako BMP zorientowany na komunikaty program odciąga kolejkę komunikatów produktu IMS , a następnie odczytuje komunikaty z kolejki produktu IBM MQ for z/OS i przetwarza je. Z kolejki komunikatów produktu IMS nie są odbierane żadne informacje. Program ponownie łączy się z menedżerem kolejek po każdym punkcie kontrolnym, ponieważ uchwyty zostały zamknięte.

W przypadku działania w BMP zorientowanej na zadanie program kontynuuje połączenie 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 dla programu 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 1283.
2. Dopasuj CLIST (CSQ4RCH1) do przykładu, aby zdefiniować położenie paneli, położenie pliku komunikatów oraz położenie modułów ładowania.

Można użyć CLIST CSQ4RCH1 , aby uruchomić zarówno wersję C, jak i wersję COBOL przykładu. Dostarczona wersja CSQ4RCH1 uruchamia wersję C i zawiera instrukcje dotyczące dostosowania, które jest niezbędne dla wersji COBOL.

Uwaga:

1. Z próbką nie udostępniono przykładowych definicji kolejek.
2. VS COBOL II nie obsługuje wielozadaniowości z ISPF, więc nie należy używać przykładowej aplikacji Message Handler po obu stronach podzielonego ekranu. Jeśli tak, to wyniki są nieprzewidywalne.

z/OS *Korzystanie z przykładu procedury obsługi komunikatów w systemie z/OS*

Po zainstalowaniu przykładu i wywołaniu go z dostosowanego CLIST CSQ4RCH1wyświetlany jest ekran pokazany w programie Rysunek 151 na stronie 1322 .

```
---------------------- 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 151. Ekran początkowy dla przykładu procedury obsługi komunikatów

Wprowadź nazwę menedżera kolejek i nazwę kolejki do wyświetlenia (rozróżnianie wielkości liter) i ekran listy komunikatów (patrz sekcja Rysunek 152 na stronie 1322).

```
---------------------- 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 152. Ekran listy komunikatów dla przykładu procedury obsługi komunikatów

Na tym ekranie wyświetlane są pierwsze 99 komunikatów w kolejce, a dla każdej z nich są wyświetlane następujące pola:

Nr komunikatu
Numer komunikatu

Data umieszczenia MM/DD/RRRR
Data umieszczenia komunikatu w kolejce (GMT)

Godzina umieszczenia GG:MM: SS
Godzina umieszczenia komunikatu w kolejce (GMT)

Nazwa formatu
MQMD.Format , pole

Identyfikator użytkownika
MQMD.UserIdentifier , pole

Typ aplikacji wstawiającej
MQMD.PutApplType , pole

Nazwa aplikacji wstawiającej
MQMD.PutApplName , pole

Wyświetlana jest także łączna liczba komunikatów w kolejce.

Na tym ekranie można wybrać komunikat, numer nie jest wyświetlany przez kursor, a następnie wyświetlany. Przykład zawiera sekcja Rysunek 153 na stronie 1323.

```
---------------------- 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 153. Zostanie wyświetlony wybrany komunikat.

Po wyświetleniu komunikatu może on zostać usunięty, pozostawiony w kolejce lub przekazany do innej kolejki. Pola `Forward to Q Mgr` i `Forward to Queue` są inicjowane wartościami z deskryptora MQMD. Te pola mogą zostać zmienione przed przekaz komunikatu.

Przykładowy projekt umożliwia wybranie i wyświetlenie tylko komunikatów z unikalnymi kombinacjami `MsgId / CorrelId` , ponieważ komunikat jest pobierany za pomocą `MsgId` i `CorrelId` jako klucza. Jeśli klucz nie jest unikalny, próba nie może pobrać wybranego komunikatu z całą pewnością.

Uwaga: Jeśli do przeglądania komunikatów używany jest przykład SCSQCLST (CSQ4RCH1), każde wywołanie powoduje zwiększenie liczby wycofań komunikatu. Jeśli chcesz zmienić zachowanie tego przykładu, skopiuj przykład i zmodyfikuj zawartość w razie potrzeby. Należy mieć świadomość, że inne aplikacje zależne od tej liczby wycofań mogą mieć wpływ na tę liczbę zwiększającą się.

z/OS *Projekt przykładowej procedury obsługi komunikatów w systemie z/OS*

W tym temacie opisano projekt każdego z programów, które składają się na przykładową aplikację procedury obsługi komunikatów.

Program do sprawdzania poprawności obiektów

Żąda ona poprawnej nazwy kolejki i menedżera kolejek.

Jeśli nie zostanie określona nazwa menedżera kolejek, zostanie użyty domyślny menedżer kolejek (jeśli jest dostępny). Można użyć tylko kolejek lokalnych. W celu sprawdzenia, czy kolejka nie jest lokalna, wysyłany jest obiekt MQINQ, który sprawdza, czy typ kolejki i błąd są lokalne. Jeśli kolejka nie została otwarta pomyślnie lub wywołanie MQGET jest zablokowane w kolejce, zwracane są komunikaty o błędach wskazujące kod powrotu CompCode i przyczyny.

Program list komunikatów

Spowoduje to wyświetlenie listy komunikatów w kolejce z informacjami o nich, takich jak putdate, puttime i format komunikatu.

Maksymalna liczba wiadomości zapisanych na liście wynosi 99. Jeśli w kolejce znajduje się więcej komunikatów niż ta, wyświetlana jest także bieżąca głębokość kolejki. Aby wybrać komunikat do wyświetlenia, należy wpisać numer komunikatu w polu wprowadzania (wartością domyślną jest 01). Jeśli pozycja nie jest poprawna, zostanie wyświetlony odpowiedni komunikat o błędzie.

Program treści wiadomości

Spowoduje to wyświetlenie treści komunikatu.

Treść jest formatowana i podzielona 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 martwe litery (MQDLH) lub nagłówek kolejki transmisji (MQXQH), są one formatowane i wyświetlane przed samym buforem.

Zanim dane buforu zostaną sformatowane, w wierszu tytułu wyświetlana jest długość buforu komunikatu w bajtach. Maksymalna wielkość buforu wynosi 32768 bajtów, a każdy komunikat dłuższy niż ten jest obcinany. Zostanie wyświetlony pełny rozmiar buforu wraz z komunikatem wskazującą, że wyświetlane są tylko pierwsze 32768 bajtów wiadomości.

Dane buforu są sformatowane na dwa sposoby:

1. Po wydrukowaniu przesunięcia w buforze dane buforu są wyświetlane w postaci szesnastkowej.
2. Dane buforu są następnie wyświetlane ponownie jako wartości EBCDIC. Jeśli nie można wydrukować żadnej wartości EBCDIC, zamiast niej drukuje się kropkę (.).

Można wprowadzić D do usunięcia lub F w celu przekazania do pola działania. Jeśli zostanie wybrana opcja przekazania komunikatu, należy poprawnie ustawić wartości `forward-to queue` i `queue manager name` . Wartości domyślne dla tych pól są odczytyane z pól ReplyTo(ReplyTo) i ReplyTo(ReplyTo) w polach QMgr.

Jeśli komunikat zostanie wyświetlony, każdy blok nagłówka zapisany w buforze zostanie pozbawiony. Jeśli komunikat zostanie pomyślnie przesłany, 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 umieszczenia asynchronicznego w systemie z/OS*

Przykładowy program umieszczania w pamięci asynchronicznej umieszcza komunikaty w kolejce przy użyciu wywołania asynchronicznego MQPUT. Przykład pobiera również informacje o statusie za pomocą wywołania MQSTAT.

Aplikacje asynchroniczne put korzystają z tych wywołań MQI:

- MQCONN
- MQOPEN
- MQPUT
- MQSTAT
- MQCLOSE
- MQDISC

Programy przykładowe są dostarczane w języku programowania C.

Aplikacje asynchroniczne umieszczone w środowisku wsadowym są uruchamiane. Więcej informacji na ten temat zawiera sekcja Inne przykłady dla aplikacji wsadowych.

Ten temat zawiera również informacje na temat projektu asynchronicznego programu Consumption i uruchamiania przykładu CSQ4BCS2 .

- "Uruchamianie przykładu CSQ4BCS2" na stronie 1325
- "Projekt przykładowego programu Asynchronicznego Put" na stronie 1325

Uruchamianie przykładu CSQ4BCS2

Ten przykładowy program przyjmuje maksymalnie sześć parametrów:

1. Nazwa kolejki docelowej (wymagane).
2. Nazwa menedżera kolejek (opcjonalnie).
3. Opcje otwierania (opcjonalne).
4. Opcje zamknięcia (opcjonalne).
5. Nazwa docelowego menedżera kolejek (opcjonalnie).
6. Nazwa kolejki dynamicznej (opcjonalnie).

Jeśli menedżer kolejek nie jest określony, CSQ4BCS2 łączy się z domyślnym menedżerem kolejek. Treść komunikatu jest udostępniana za pośrednictwem standardowego wejścia (**SYSIN DD**).

Istnieje przykładowy skrypt JCL do uruchomienia programu, który znajduje się w katalogu CSQ4BCSP.

Projekt przykładowego programu Asynchronicznego Put

Program używa wywołania MQOPEN z dostarczonym opcjami wyjściowymi lub z opcjami MQOO_OUTPUT i MQOO_FAIL_IF_QUIESCING, aby otworzyć kolejkę docelową w celu umieszczania komunikatów.

Jeśli program nie może otworzyć kolejki, program wyprowadza komunikat o błędzie zawierający kod przyczyny zwrócony przez wywołanie MQOPEN. Aby program był prosty w tym i kolejnych wywołaniach MQI, dla wielu z nich używane są wartości domyślne.

Dla każdego wiersza danych wejściowych program odczytuje tekst do buforu i korzysta z wywołania MQPUT z opcją MQPMO_ASYNC_RESPONSE w celu utworzenia komunikatu datagramu zawierającego tekst tego wiersza i asynchronicznie umieszcza komunikat w kolejce docelowej. Program będzie kontynuowany do momentu osiągnięcia końca danych wejściowych lub do momentu, gdy wywołanie MQPUT nie powiedzie się. Jeśli program dociera do końca danych wejściowych, zamyka kolejkę za pomocą wywołania MQCLOSE.

Następnie program wysyła wywołanie MQSTAT, które zwraca strukturę MQSTS, a następnie wyświetla komunikaty zawierające liczbę pomyślnie umieszczonych komunikatów, liczbę komunikatów wysłanych z ostrzeżeniem oraz liczbę niepowodzeń.

Uwaga: Aby obserwować, co się dzieje, gdy błąd MQPUT jest wykrywany przez wywołanie MQSTAT, ustaw wartość MAXDEPTH w kolejce docelowej na małą wartość.

z/OS *Przykład asynchronicznego wykorzystania zadania wsadowego w systemie z/OS*

Przykładowy program CSQ4BCS1 jest dostarczany w języku C, który demonstruje użycie obiektów MQCB i MQCTL w celu asynchronicznego korzystania z komunikatów z wielu kolejek.

Przykłady wykorzystania asynchronicznego są uruchamiane w środowisku wsadowym. Więcej informacji na ten temat zawiera sekcja Inne przykłady dla aplikacji wsadowych.

Istnieje również przykład COBOL, który działa w środowisku CICS , patrz “Przykład CICS Asynchroniczna konsumpcja i Publikowanie/subskrypcja w systemie z/OS” na stronie 1327.

Aplikacje korzystają z następujących wywołań MQI:

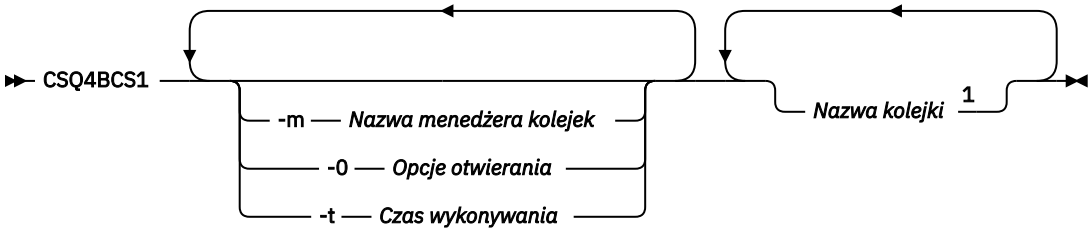
- MQCONN
- MQOPEN
- MQCLOSE
- MQDISC
- MQCB
- Komenda MQCTL

W tym temacie przedstawiono również informacje na temat następujących nagłówków:

- “Uruchamianie przykładu CSQ4BCS1” na stronie 1326
- “Projekt przykładowego programu kontroli zużycia asynchronicznego zadania wsadowego” na stronie 1327

Uruchamianie przykładu CSQ4BCS1

Ten przykładowy program jest zgodny z następującą składnią:



Uwagi:

[1] Ten przykład obsługuje maksymalną liczbę 10 kolejek. Podaj więcej niż jedną nazwę kolejki w celu odczytania komunikatów z wielu kolejek.

Istnieje przykładowy skrypt JCL do uruchomienia tego programu, który znajduje się w katalogu CSQ4BCSC.

Projekt przykładowego programu kontroli zużycia asynchronicznego zadania wsadowego

W przykładzie pokazano, jak odczytywać komunikaty z wielu kolejek w kolejności ich przybycia. Wymagałoby to więcej kodu przy użyciu synchronicznego wywołania MQGET. W przypadku wykorzystania asynchronicznego nie jest wymagane odpytywanie, a zarządzanie wątkami i pamięcią masową jest wykonywane przez produkt IBM MQ. W przykładowym programie błędy są zapisywane w konsoli.

Przykładowy kod składa się z następujących kroków:

1. Zdefiniuj funkcję wywołania zwrotnego pojedynczego wykorzystania komunikatów.

```
void MessageConsumer(MQHCONN   hConn,
MQMD  * pMsgDesc,
MQGMO  * pGetMsgOpts,
MQBYTE * Buffer,
MQCBC  * pContext)
{ ... }
```

2. Połącz się 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);
```

   `cbd.CallbackFunction` nie musi być ustawiany dla każdej kolejki; jest to pole tylko wejściowe. Z każdą kolejką można powiązać inną funkcję zwrotną.

4. Rozpoczęcie korzystania z komunikatów.

```
MQCTL(Hcon,MQOP_START,&ctlo,&CompCode,&Reason);
```

5. Poczekaj, aż użytkownik naciska klawisz Enter, a następnie zaprzestanie korzystania z komunikatów.

```
MQCTL(Hcon,MQOP_STOP,&ctlo,&CompCode,&Reason);
```

6. Na koniec odłącz się od menedżera kolejek.

```
MQDISC(&Hcon,&CompCode,&Reason);
```

z/OS *Przykład CICS Asynchroniczna konsumpcja i Publikowanie/subskrypcja w systemie z/OS*

Programy przykładowe Zużycie asynchroniczne i Publikowanie/Subskrybuj demonstrują użycie asynchronicznego wykorzystania oraz funkcje publikowania i subskrybowania w produkcie CICS.

Program *Klient rejestracji* rejestruje trzy procedury obsługi wywołania zwrotnego (procedura obsługi zdarzeń i dwa konsumenty komunikatów) i rozpoczyna asynchroniczną konsumpcję. Program *Klient przesyłania komunikatów* umieszcza komunikaty w kolejce lub publikuje odpowiednie komunikaty z poziomu konsoli produktu 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 za pomocą komunikatów, które otrzymuje, w celu SUSPEND, RESUME lub DEREGISTER dowolnego z procedur obsługi wywołania zwrotnego. Można go również użyć do wydania komendy MQCTL STOP w celu zakończenia operacji asynchronicznego wykorzystania

w elemencie sterującym. Inny konsument komunikatów jest zarejestrowany w celu zasubskrybowania tematu.

Każdy program wydaje instrukcje języka COBOL DISPLAY w odpowiednich punktach w celu wyświetlenia zachowania przykładu.

Aplikacje korzystają z następujących wywołań MQI:

- MQOPEN
- MQPUT
- MQSUB
- MQGET
- MQCLOSE
- MQCB
- Komenda MQCTL

Programy są dostarczane w języku COBOL. Informacje na temat aplikacji CICS można znaleźć w sekcji CICS Asynchronous Consumption and Publish/Subscribe samples .

W tym temacie znajdują się również następujące informacje:

- “Konfiguracja” na stronie 1328
- “Klient rejestracji CSQ4CVRG” na stronie 1328
- “Procedura obsługi zdarzeń CSQ4CVEV” na stronie 1329
- “Prosty konsument komunikatów CSQ4CVCN” na stronie 1329
- “Konsument komunikatów sterowania CSQ4CVCT” na stronie 1329
- “Klient przesyłania komunikatów CSQ4CVPT” na stronie 1329

Konfiguracja

Nazwy kolejki i tematu używane przez konsumentów komunikatów są zakodowane w programach dotyczących rejestracji i przesyłania komunikatów.

Kolejka **SAMPLE.CONTROL.QUEUE**, przed uruchomieniem przykładu, należy zdefiniować w menedżerze kolejek powiązanym z regionem CICS . Temat **Wiadomości/media/Filmy**może zostać zdefiniowany, jeśli jest wymagany, lub jest tworzony w czasie wykonywania w domyślnym obiekcie administracyjnym, jeśli nie istnieje.

Programy CICS i definicje transakcji mogą być instalowane przez instalację grupy: CSQ4SAMP.

Klient rejestracji CSQ4CVRG

Program kliencki rejestracji musi być uruchomiony w ramach transakcji CICS MVRG. Nie ma danych wejściowych.

Po uruchomieniu klient rejestracji rejestruje następujące procedury obsługi wywołania zwrotnego za pomocą obiektu MQCB:

- CSQ4CVEV jako procedura obsługi zdarzeń.
- CSQ4CVCN jako konsument komunikatów w temacie **News/Media/Movies**(Wiadomości/Media/Filmy).
- CSQ4CVCT jako konsument komunikatów w kolejce, **SAMPLE.CONTROL.QUEUE**.

Klient rejestracji przekazuje strukturę danych zawierającą nazwy wszystkich trzech zarejestrowanych procedur obsługi Callback 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 wykorzystania asynchronicznego i zawieś do momentu zwrócenia do

niego elementu sterującego (na przykład przez jeden z procedur obsługi wywołania zwrotnego, które wydały komendę MQCTL STOP).

Procedura obsługi zdarzeń CSQ4CVEV

Po uruchomieniu procedura obsługi zdarzeń wyświetla komunikat informujący o typie wywołania (na przykład START). W przypadku pracy z kodem przyczyny produktu IBM MQ CONNECTION_QUIESCING, procedura obsługi zdarzeń wysyła komendę MQCTL STOP, aby zakończyć asynchroniczne wykorzystanie i zwracanie sterowania do klienta rejestracji.

Prosty konsument komunikatów CSQ4CVCN

W przypadku pracy z tym konsumentem komunikatów wyświetlany jest komunikat wskazujący typ wywołania (np. REGISTER). W przypadku pracy z typem wywołania MSG_REMOVED konsument komunikatów pobiera komunikat przychodzący i wyprowadza go do protokołu zadania produktu CICS .

Konsument komunikatów sterowania CSQ4CVCT

W przypadku uruchomienia tego komunikatu konsument komunikatów wyświetla komunikat wskazujący typ wywołania (na przykład START). W przypadku pracy z typem wywołania MSG_REMOVED konsument komunikatów pobiera komunikat przychodzący oraz strukturę danych przekazanej przez klienta rejestracji. Na podstawie treści komunikatu wydaje odpowiednie komendy MQCB lub MQCTL, aby wykonać jedną z następujących czynności:

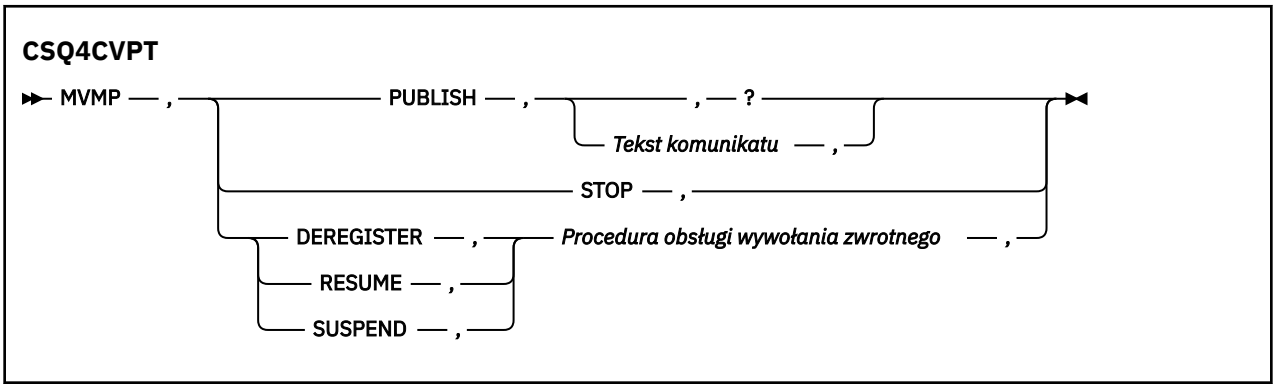
- STOP Asynchroniczny konsumpcja (zwracanie kontroli do Klienta Rejestracji).
- SUSPEND, RESUME lub DEREGISTER nazwana procedura obsługi Callback (w tym sama nazwa).

Klient przesyłania komunikatów CSQ4CVPT

Klient przesyłania komunikatów ma dwie funkcje:

- Publikuje on komunikat do tematu dotyczącego wykorzystania przez konsumenta komunikatów CSQ4CVCN.
- Umieszcza komunikat sterujący w kolejce do wykorzystania przez konsument komunikatów sterujących CSQ4CVCT, co powoduje potencjalną zmianę w działaniu przykładu.

Program Klient przesyłania komunikatów musi być uruchomiony z poziomu konsoli produktu CICS w ramach transakcji produktu CICS i pobiera dane wejściowe wiersza komend z następującą składnią:



PUBLISH
Opublikuj tekst komunikatu (lub komunikat domyślny) jako zatrzymany komunikat do wykorzystania przez konsumenta prostego komunikatu.

STOP
Zatrzymaj Asynchroniczną Konsumpcję.

DEREGISTER
Wyrejestruj nazwaną procedurę obsługi Callback.

RESUME
Wznów nazwaną procedurę obsługi Callback.

SUSPEND
Zawieś nazwaną procedurę obsługi Callback.

Pola wejściowe są pozycyjne i rozdzielane przecinkami. W słowach kluczowych i nazwach procedur obsługi wywołania zwrotnego nie jest rozróżniana wielkość liter.

Przykłady:

Tabela 193. Przykłady danych wejściowych

Przykład	**Opis**
MVMP, PUBLIKUJ,,	Opublikuj komunikat domyślny
MVMP,publish, `A short message`,	Opublikuj dany tekst
MVMP, STOP,	Zatrzymaj wykorzystanie asynchroniczne
MVMP,DEREGISTER,CSQ4CVEV,	Wyrejestruj procedurę obsługi zdarzeń
MVMP,resume,csq4cvcn,	Wznów prosty konsument komunikatów
MVMP,SUSPEND,CSQ4CVEV,	Zawieś procedurę obsługi zdarzeń

Gdzie MVMP jest transakcją CICS powiązaną z programem przesyłania komunikatów CSQ4CVPT.

Uwaga:

- Zawieszenie lub wyrejestrowywanie wszystkich procedur obsługi wywołania zwrotnego kończy działanie START_WAIT wystawione przez klienta rejestracji, zwracając do niego element sterujący i kończąc zadanie.
- Zawieszenie lub wyrejestrowywanie procedury obsługi wywołania zwrotnego elementu sterującego celowo nie zostało uniemożliwione, ale usuwa możliwość dalszej kontroli zachowania próby.

z/OS *Przykład publikowania/subskrypcji w systemie z/OS*

Przykładowe programy publikowania/subskrypcji demonstrują sposób użycia funkcji publikowania i subskrypcji w produkcie IBM MQ.

Istnieją cztery programy przykładowe języka programowania w języku C i dwóch językach COBOL demonstruje sposób programu do interfejsu publikowania/subskrybowania produktu IBM MQ . Programy są dostarczane w języku C i COBOL. Aplikacje działają w środowisku wsadowym. Patrz sekcja Przykłady publikowania/subskrypcji dla aplikacji wsadowych.

Istnieją również przykłady języka COBOL, które działają w środowisku produktu CICS . Patrz sekcja “Przykład CICS Asynchroniczna konsumpcja i Publikowanie/subskrypcja w systemie z/OS” na stronie 1327.

Ten temat zawiera również informacje na temat uruchamiania przykładowych programów publikowania/subskrypcji. Te przykładowe programy zawierają:

- “Uruchamianie przykładu CSQ4BCP1” na stronie 1331
- “Uruchamianie przykładu CSQ4BCP2” na stronie 1331
- “Uruchamianie przykładu CSQ4BCP3” na stronie 1331
- “Uruchamianie przykładu CSQ4BCP4” na stronie 1331
- “Uruchamianie przykładu CSQ4BVP1” na stronie 1332
- “Uruchamianie przykładu CSQ4BVP2” na stronie 1332

Uruchamianie przykładu CSQ4BCP1

Ten program jest napisany w języku C, który publikuje komunikaty w temacie. Przed uruchomieniem tego programu należy uruchomić jedną z przykładowych subskrybentów.

Ten program przyjmuje maksymalnie cztery parametry:

1. Nazwa docelowego łańcucha tematu (wymagane).
2. Nazwa menedżera kolejek (opcjonalnie).
3. Opcje otwierania (opcjonalne).
4. Opcje zamknięcia (opcjonalne).

Jeśli menedżer kolejek nie jest określony, CSQ4BCP1 łączy się z domyślnym menedżerem kolejek. Istnieje przykładowy skrypt JCL do uruchomienia programu, który znajduje się w katalogu CSQ4BCPP.

Treść komunikatu jest udostępniana za pośrednictwem standardowego wejścia (**SYSIN DD**).

Uruchamianie przykładu CSQ4BCP2

Ten program jest napisany w języku C, subskrybuje wątek i drukuje otrzymane wiadomości.

Ten program przyjmuje maksymalnie trzy parametry:

1. Nazwa docelowego łańcucha tematu (wymagane).
2. Nazwa menedżera kolejek (opcjonalnie).
3. Opcje subskrypcji MQSD (opcjonalne).

Jeśli menedżer kolejek nie jest określony, CSQ4BCP2 łączy się z domyślnym menedżerem kolejek. Istnieje przykładowy skrypt JCL do uruchomienia programu, który znajduje się w katalogu CSQ4BCPS.

Uruchamianie przykładu CSQ4BCP3

Ten program jest napisany w języku C; subskrybuje wątek przy użyciu określonej przez użytkownika kolejki docelowej i drukuje otrzymane komunikaty.

Ten program przyjmuje maksymalnie cztery parametry:

1. Nazwa docelowego łańcucha tematu (wymagane).
2. Nazwa miejsca docelowego (wymagane).
3. Nazwa menedżera kolejek (opcjonalnie).
4. Opcje subskrypcji MQSD (opcjonalne).

Jeśli menedżer kolejek nie jest określony, CSQ4BCP3 łączy się z domyślnym menedżerem kolejek. Istnieje przykładowy skrypt JCL do uruchomienia 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 te dostępne na prostszy przykład MQSUB: CSQ4BCP2. Oprócz ładunku komunikatu właściwości komunikatu dla każdego komunikatu są odbierane i wyświetlane.

Ten program przyjmuje zmienną set parametrów:

- **-t** *Topic string* (wymagane).
- **-o** *Topic object name* (wymagane).
- **-m** *Queue manager name* (opcjonalnie).
- **-q** *Destination queue name* (opcjonalnie).
- **-w** *Wait interval on MQGET in seconds* (opcjonalnie), gdzie *sekundy* mogą mieć dowolną z następujących wartości:

  - `unlimited`: MQWI_UNLIMITED
  - `none`: Brak 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 produktu MQCLOSE.

Jeśli menedżer kolejek nie jest określony, CSQ4BCP4 łączy się z domyślnym menedżerem kolejek. Istnieje przykładowy skrypt JCL do uruchomienia programu, który znajduje się w katalogu CSQ4BCPE.

Uruchamianie przykładu CSQ4BVP1

Ten program jest napisany w języku COBOL, publikujący komunikaty do tematu. Przed uruchomieniem tego programu należy uruchomić jedną z przykładowych subskrybentów.

Ten program nie przyjmuje żadnych parametrów. Produkt **SYSIN DD** udostępnia nazwę tematu wejściowego, nazwę menedżera kolejek i treść komunikatu.

Jeśli menedżer kolejek nie jest określony, CSQ4BVP1 łączy się z domyślnym menedżerem kolejek. Istnieje przykładowy skrypt JCL do uruchomienia programu, który znajduje się w katalogu CSQ4BVPP.

Uruchamianie przykładu CSQ4BVP2

Ten program jest napisany w języku COBOL, subskrybuje wątek i drukuje otrzymane wiadomości.

Ten program nie przyjmuje żadnych parametrów. Produkt **SYSIN DD** udostępnia dane wejściowe dla nazwy tematu i nazwy menedżera kolejek.

Jeśli menedżer kolejek nie jest określony, CSQ4BVP1 łączy się z domyślnym menedżerem kolejek. Istnieje przykładowy skrypt JCL do uruchomienia programu, który znajduje się w katalogu CSQ4BVPP.

z/OS *Przykład właściwości Set and Inquire message w systemie z/OS*

Przykładowe programy właściwości komunikatu demonstrują dodawanie właściwości zdefiniowanych przez użytkownika do uchwytu komunikatu i zapytanie o właściwości powiązane z tym komunikatem.

Aplikacje korzystają z następujących wywołań MQI:

- MQCONN
- MQOPEN
- MQPUT
- MQGET
- MQCLOSE
- MQDISC
- MQCRTMH
- MQDLTMH
- MQINQMP
- MQSETMP

Programy są dostarczane w języku C. Aplikacje działają w środowisku wsadowym. Więcej informacji na ten temat zawiera sekcja Inne przykłady dla aplikacji wsadowych.

Program CSQ4BCM1 służy do uzyskiwania informacji o właściwościach uchwytu komunikatu z kolejki komunikatów. Jest to przykład użycia wywołania API MQINQMP. Próbka pobiera jeden komunikat z kolejki, a następnie drukuje wszystkie właściwości uchwytu komunikatu.

Program CSQ4BCM2 służy do ustawiania właściwości uchwytu komunikatu w kolejce komunikatów. Jest to przykład użycia wywołania funkcji API MQSETMP. W tym przykładzie tworzony jest uchwyt komunikatu i umieszcza go w polu `MsgHandle` struktury MQGMO. Następnie umieszcza komunikat w kolejce.

Inne przykłady uzyskiwania informacji i drukowania właściwości komunikatów są zawarte w przykładowych programach CSQ4BCG1 i CSQ4BCP4 .

Ten temat zawiera również informacje na temat uruchamiania przykładów właściwości komunikatów Set i Inquire pod następującymi nagłówkami:

- “Uruchamianie przykładu CSQ4BCM1” na stronie 1333
- “Uruchamianie przykładu CSQ4BCM2” na stronie 1333

Uruchamianie przykładu CSQ4BCM1

Ten program przyjmuje maksymalnie cztery parametry:

1. Nazwa kolejki docelowej (wymagane).
2. Nazwa menedżera kolejek (opcjonalnie).
3. Opcje otwierania (opcjonalne).
4. Opcje zamknięcia (opcjonalne).

Uruchamianie przykładu CSQ4BCM2

Ten program przyjmuje maksymalnie sześć parametrów:

1. Nazwa kolejki docelowej (wymagane).
2. Nazwa menedżera kolejek (opcjonalnie).
3. Opcje otwierania (opcjonalne).
4. Opcje zamknięcia (opcjonalne).
5. Nazwa docelowego menedżera kolejek (opcjonalnie).
6. Nazwa kolejki dynamicznej (opcjonalnie).

Nazwy właściwości, wartości i treść komunikatu są udostępniane za pośrednictwem standardowego wejścia (**SYSIN DD**). Istnieje przykładowy skrypt JCL do uruchomienia programu, który znajduje się w katalogu CSQ4BCMP.

Tworzenie aplikacji dla składnika Managed File Transfer

Określ programy, które mają być uruchamiane z programem Managed File Transfer, z Apache Ant , z Managed File Transfer, z programami wyjściowymi użytkownika Managed File Transfer sterowaniem Managed File Transfer , przez umieszczanie komunikatów w kolejce komend agenta.

Określanie programów do uruchomienia za pomocą programu MFT

Programy można uruchamiać w systemie, w którym działa produkt Managed File Transfer Agent . W ramach żądania przesyłania plików można określić program, który ma być uruchamiany przed rozpoczęciem przesyłania lub po jego zakończeniu. Dodatkowo można uruchomić program, który nie jest częścią żądania przesyłania plików, wysyłając żądanie wywołania zarządzanego.

Istnieje pięć scenariuszy, w których można określić program do uruchomienia:

- W ramach żądania transferu, w agencie źródłowym, przed rozpoczęciem przesyłania
- W ramach żądania transferu, 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. Istnieje możliwość wysłania żądania do agenta w celu uruchomienia programu. Ten scenariusz jest czasem określany jako wywołanie zarządzane.

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. **DestinationTransferEndExits** jest uruchamiany tylko wtedy, gdy operacja przesyłania zakończy się pomyślnie lub częściowo pomyślnie.
2. **postDestinationCall** jest uruchamiany tylko wtedy, gdy operacja przesyłania zakończy się pomyślnie lub częściowo pomyślnie.
3. **SourceTransferEndExits** jest uruchamiany dla pomyślnych, częściowo zakończonych powodzeniem lub zakończonych niepowodzeniem transferów.
4. Program **postSourceCall** jest wywoływany tylko wtedy, gdy:
   - Przesyłanie nie zostało anulowane.
   - Wynik jest pomyślny lub częściowo pomyślny.
   - Wszystkie programy przesyłania po miejscu docelowym przebiegły pomyślnie.

Istnieje kilka sposobów określenia programu, który ma zostać uruchomiony. Są to następujące opcje:

Użyj zadania Apache Ant

Aby uruchomić program, należy użyć jednej z zadań `fte:filecopy`, `fte:filemove`i `fte:call` Ant . Za pomocą zadania Ant można określić program w jednym z pięciu scenariuszy, korzystając z zagnieżdżonych elementów `fte:presrc`, `fte:predst`, `fte:postdst`, `fte:postsrc`i `fte:command` . Więcej informacji na ten temat zawiera sekcja Zagnieżdżone elementy wywołania programu.

Edytuj komunikat żądania przesyłania plików

Istnieje możliwość edytowania kodu XML, który jest generowany przez żądanie przesyłania. Za pomocą tej metody można uruchomić program w jednym 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 w przypadku parametru **fteCreateTransfer -td** . Więcej informacji na ten temat zawiera sekcja Przykłady komunikatów żądania wywołania agenta MFT.

Użyj komendy `fteCreateTransfer`

Aby określić programy 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 połączenia zarządzanego. Informacje na temat parametrów, które mają być używane, zawiera sekcja **fteCreateTransfer**: uruchomienie nowego przesyłania plików. Przykłady użycia tej komendy można znaleźć w sekcji Przykłady użycia 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.

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 trakcie odtwarzania, dwie źródłowe szczeliny przesyłania są zajęte do momentu zakończenia przesyłania zarządzanego, anulowania lub przekroczenia limitu czasu z powodu **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 produktu Apache Ant z produktem MFT

Program Managed File Transfer udostępnia zadania, których można użyć do zintegrowania funkcji przesyłania plików z narzędziem Apache Ant .

Za pomocą komendy **fteAnt** można uruchamiać zadania Ant w środowisku Managed File Transfer , które zostało już skonfigurowane. Zadania przesyłania plików Ant można użyć ze skryptów produktu Ant w celu koordynowania złożonych operacji przesyłania plików z interpretowanego języka skryptowego.

Więcej informacji na temat produktu Apache Antmożna znaleźć na stronie WWW projektu produktu Apache Ant : https://ant.apache.org/

Pojęcia pokrewne

“Pierwsze kroki z użyciem skryptów Ant przy użyciu produktu MFT” na stronie 1336
Użycie skryptów Ant z programem Managed File Transfer umożliwia skoordynowanie złożonych operacji przesyłania plików z interpretowanego języka skryptowego.

Odsyłacze pokrewne

fteAnt: uruchamianie zadań programu Ant w produkcie MFT

“Przykładowe zadania Ant dla MFT” na stronie 1337
Istnieje wiele przykładowych skryptów Ant udostępnionych z instalacją produktu Managed File Transfer. Przykłady te znajdują się w katalogu *MQ_INSTALLATION_PATH*/mqft/samples/fteant. Każdy przykładowy skrypt zawiera element docelowy produktu `init` , a następnie zmodyfikuj właściwości ustawione w celu `init` , aby uruchomić te skrypty z konfiguracją użytkownika.

Pierwsze kroki z użyciem skryptów Ant przy użyciu produktu MFT

Użycie skryptów Ant z programem Managed File Transfer umożliwia skoordynowanie złożonych operacji przesyłania plików z interpretowanego języka skryptowego.

Skrypty Ant

Skrypty Ant (lub pliki budowania) to dokumenty XML definiujące co najmniej jeden element docelowy. Te cele zawierają elementy zadania do uruchomienia. Program Managed File Transfer udostępnia zadania, których można użyć do zintegrowania funkcji przesyłania plików z produktem Apache Ant. Informacje na temat skryptów produktu Ant można znaleźć na stronie WWW projektu produktu Apache Ant : https://ant.apache.org/

Przykłady skryptów produktu Ant , które korzystają z zadań programu Managed File Transfer , są dostarczane wraz 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 przesyłania plików Ant od innych zadań produktu Ant , które mogą współużytkować tę samą nazwę. Przestrzeń nazw definiuje się 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 program Ant , aby poszukiwać definicji zadań z przedrostkiem `fte` , które znajdują się w bibliotece `com.ibm.wmqfte.ant.taskdefs`.

Nie ma potrzeby używania produktu `fte` jako przedrostka przestrzeni nazw. Można użyć dowolnej wartości. Przedrostek przestrzeni nazw `fte` jest używany we wszystkich przykładach i przykładowych skryptach produktu Ant .

Uruchamianie skryptów produktu Ant

Aby uruchomić skrypty 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 programie 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 dla produktu MFT.

Odsyłacze pokrewne

fteAnt: uruchamianie zadań programu Ant w produkcie MFT

“Przykładowe zadania Ant dla MFT” na stronie 1337

Istnieje wiele przykładowych skryptów Ant udostępnionych z instalacją produktu Managed File Transfer. Przykłady te znajdują się w katalogu *MQ_INSTALLATION_PATH*`/mqft/samples/fteant`. Każdy przykładowy skrypt zawiera element docelowy produktu `init` , a następnie zmodyfikuj właściwości ustawione w celu `init` , aby uruchomić te skrypty z konfiguracją użytkownika.

Przykładowe zadania Ant dla MFT

Istnieje wiele przykładowych skryptów Ant udostępnionych z instalacją produktu Managed File Transfer. Przykłady te znajdują się w katalogu *MQ_INSTALLATION_PATH*`/mqft/samples/fteant`. Każdy przykładowy skrypt zawiera element docelowy produktu `init` , a następnie zmodyfikuj właściwości ustawione w celu `init` , aby uruchomić te skrypty z konfiguracją użytkownika.

e-mail

W przykładzie e-mail przedstawiono sposób użycia zadań programu Ant w celu przesłania pliku i wysłania wiadomości e-mail na podany adres e-mail, jeśli operacja przesyłania nie powiedzie się. Skrypt sprawdza, czy agenty źródłowe i docelowe są aktywne i mogą przetwarzać transfery, używając zadania Managed File Transfer ping . Jeśli oba agenty są aktywne, skrypt korzysta z zadania Managed File Transfer fte: filecopy w celu przesłania pliku między agentami źródłowymi i docelowym, bez usuwania oryginalnego pliku. Jeśli operacja przesyłania nie powiedzie się, skrypt wysyła wiadomość e-mail zawierającą informacje o niepowodzeniu, korzystając ze standardowego zadania Ant e-mail .

koncentrator

The koncentrator sample is made up of two scripts: `hubcopy.xml` and `hubprocess.xml.` The `hubcopy.xml` script shows how you can use Ant scripting to build 'hub and spoke' style topologies. W tym przykładzie dwa pliki są przesyłane z agentów działających na komputerach z gadami do agenta działającego na komputerze centralnym. Oba pliki są przesyłane w tym samym czasie, a po zakończeniu przesyłania skrypt `hubprocess.xml` Ant jest uruchamiany na komputerze centralnym w celu przetwarzania plików. Jeśli oba pliki są przesyłane poprawnie, skrypt Ant konkatenuje zawartość tych plików. Jeśli pliki nie zostaną przesłane poprawnie, skrypt Ant będzie czyści przez usunięcie wszystkich przesłanych danych. Aby ten przykład działał poprawnie, należy umieścić skrypt `hubprocess.xml` w ścieżce komend agenta koncentratora. Więcej informacji na temat ustawiania ścieżki komend dla agenta zawiera sekcja commandPath MFT property.

librarytransfer (tylko platformaIBM i)

IBM i

IBM i W przykładzie biblioteki transferowej zademonstrować sposób użycia zadań Ant do przesyłania biblioteki IBM i w jednym systemie IBM i do drugiego systemu IBM i .

IBM i Produkt IBM WebSphere MQ File Transfer Edition 7.0.2 w systemie IBM i nie zawiera bezpośredniego wsparcia dla transferów rodzimych obiektów biblioteki produktu IBM i . W przykładzie z transferem biblioteki używana jest obsługa rodzimego zbioru składowania w systemie IBM i z predefiniowanymi zadaniami Ant dostępnymi w produkcie Managed File Transfer w celu przesyłania rodzimych obiektów biblioteki między dwoma systemami IBM i . W przykładzie użyto zagnieżdżonego elementu < presrc> w zadaniu Managed File Transfer filecopy w celu wywołania skryptu wykonywalnego `librarysave.sh` , który zapisuje żądaną bibliotekę w systemie agenta źródłowego w tymczasowy zbiór składowania. Plik 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 do systemu docelowego.

IBM i Przed uruchomieniem tego przykładu należy wykonać określoną konfigurację w sposób opisany w pliku `librarytransfer.xml` . Użytkownik musi mieć również działające środowisko Managed File Transfer na dwóch komputerach IBM i . Konfiguracja musi składać się z agenta źródłowego działającego na pierwszym komputerze IBM i i agenta docelowego działającego na drugim komputerze z produktem IBM i . Te dwa agenty muszą być w stanie komunikować się ze sobą.

IBM i Przykład biblioteki transferowej składa się z następujących trzech plików:

- `librarytransfer.xml`
- `librarysave.sh` (< presrc> skrypt wykonywalny)
- `librayrestore.sh` (< postdst> skrypt wykonywalny)

Przykładowe pliki znajdują się w następującym katalogu: `/QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/librarytransfer`

IBM i Aby uruchomić tę próbkę, użytkownik musi wykonać następujące kroki:

1. Uruchom sesję Qshell. W oknie komend IBM i wpisz: STRQSH
2. Zmień katalog na katalog `bin` w następujący sposób:

```
cd  /QIBM/ProdData/WMQFTE/V7/bin
```

3. Po wykonaniu wymaganej konfiguracji uruchom próbę za pomocą następującej komendy:

```
fteant -f /QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/librarytransfer/librarytransfer.xml
```

physicalfiletransfer (tylko platformaIBM i)

IBM i Przykład fizjcalfiletransfer demonstruje sposób użycia zadań programu Ant do przesyłania źródłowego zbioru fizycznego lub zbioru bazy danych z biblioteki w jednym systemie IBM i do biblioteki w drugim systemie IBM i .

IBM i Produkt IBM WebSphere MQ File Transfer Edition 7.0.2 w systemie IBM i nie zawiera bezpośredniego wsparcia dla transferów rodzimych źródłowych zbiorów fizycznych lub plików bazy danych w systemie IBM i . W próbce physicalfiletransfer używana jest rodzima obsługa zbiorów składowania w produkcie IBM i z predefiniowanymi zadaniami Ant dostępnymi w produkcie Managed File Transfer w celu przesłania kompletnych plików fizycznych i zbiorów bazy danych między dwoma systemami IBM i . W przykładzie użyto elementu zagnieżdżonego < presrc> zagnieżdżonego w zadaniu

Managed File Transfer filecopy w celu wywołania skryptu wykonywalnego `physicalfilesave.sh` w celu zapisania żądanego źródłowego zbioru fizycznego lub zbioru bazy danych z biblioteki w źródłowym systemie agenta w tymczasowy zbiór składowania. Plik składowania 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 zbioru składowania do określonej biblioteki w systemie docelowym.

IBM i Przed uruchomieniem tego przykładu należy wykonać określoną konfigurację w sposób opisany w pliku `physicalfiletransfer.xml` . Użytkownik musi mieć 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 . Te dwa agenty muszą być w stanie komunikować się ze sobą.

IBM i Próbka fizykfiletransfer składa się z następujących trzech plików:

- `physicalfiletransfer.xml`
- `physicalfilesave.sh` (< presrc> skrypt wykonywalny)
- `physicalfilerestore.sh` (< postdst> skrypt wykonywalny)

Przykładowe pliki znajdują się w następującym katalogu: `/QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/physicalfiletransfer`

IBM i Aby uruchomić tę próbkę, użytkownik musi wykonać następujące kroki:

1. Uruchom sesję Qshell. W oknie komend IBM i wpisz: STRQSH
2. Zmień katalog na katalog `bin` w następujący sposób:

```
cd  /QIBM/ProdData/WMQFTE/V7/bin
```

3. Po wykonaniu wymaganej konfiguracji uruchom próbę za pomocą następującej komendy:

```
fteant -f /QIBM/ProdData/WMQFTE/V7/samples/fteant/ibmi/physicalfiletransfer/
physicalfiletransfer.xml
```

limit czasu

Przykładowy limit czasu demonstruje sposób użycia zadań Ant do próby przesłania pliku i anulowania transferu, jeśli czas ten trwa dłużej niż określona wartość limitu czasu. Skrypt inicjuje przesyłanie plików za pomocą zadania Managed File Transfer fte: filecopy . Wynik tego przesunięcia jest odroczony. Skrypt korzysta z zadania Managed File Transfer fte: awaitoutcome Ant task , aby zaczekać określoną liczbę sekund na zakończenie przesyłania. Jeśli operacja przesyłania nie zostanie zakończona w podanym czasie, zostanie użyta opcja Managed File Transfer fte: cancel Ant task , która anuluje przesyłanie plików.

vsamtransfer

z/OS

z/OS Przykład vsamtransfer demonstruje sposób użycia zadań programu Ant do przesyłania danych z zestawu danych VSAM do innego zestawu danych VSAM za pomocą programu Managed File Transfer. Produkt Managed File Transfer obecnie nie obsługuje przesyłania zestawów danych VSAM. Przykładowy skrypt wyładowuje rekordy danych VSAM do sekwencyjnego zestawu danych za pomocą zagnieżdżonych elementów wywołania programu programu `presrc` w celu wywołania pliku wykonywalnego `datasetcopy.sh`. Skrypt korzysta z zadania Managed File Transfer fte: filemove w celu przesłania zestawu danych sekwencyjnych z agenta źródłowego do agenta docelowego. Następnie skrypt korzysta z elementów zagnieżdżonych wywołania programu programu `postdst` w celu wywołania skryptu `loadvsam.jcl` . Ten skrypt JCL ładuje przesłane rekordy zestawu danych do docelowego zestawu danych VSAM. W tym przykładzie używane jest zadanie JCL dla wywołania docelowego w celu

zademonstrowania tej opcji językowej. Ten sam wynik można również osiągnąć za pomocą drugiego skryptu powłoki.

z/OS Ten przykład nie wymaga, aby zbiory danych źródłowych i docelowych były VSAM. Przykład działa dla każdego zestawu danych, jeśli źródłowy i docelowy zestaw danych są tego samego typu.

z/OS Aby ten przykład mógł działać poprawnie, należy umieścić skrypt `datasetcopy.sh` w ścieżce komend agenta źródłowego i skryptu `loadvsam.jcl` w ścieżce komend agenta docelowego. Więcej informacji na temat ustawiania ścieżki komend dla agenta zawiera sekcja commandPath MFT property.

zip

Przykładowy plik zip składa się z dwóch skryptów: `zip.xml` i `zipfiles.xml`. W przykładzie przedstawiono sposób użycia elementu zagnieżdżonego produktu `presrc` w zadaniu Managed File Transfer fte: filemove w celu uruchomienia skryptu Ant przed wykonaniem operacji przenoszenia pliku. Skrypt `zipfiles.xml` wywoływany przez zagnieżdżony element `presrc` w skrypcie `zip.xml` kompresuje zawartość katalogu. Skrypt `zip.xml` przesyła skompresowany plik. W tym przykładzie wymagane jest, aby skrypt `zipfiles.xml` Ant był obecny w ścieżce komend agenta źródłowego. Jest to spowodowane tym, że skrypt `zipfiles.xml` Ant zawiera element docelowy używany do kompresowania zawartości katalogu w agencie źródłowym. Więcej informacji na temat ustawiania ścieżki komend dla agenta zawiera sekcja commandPath MFT property.

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 194. 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. **DestinationTransferEndExits** jest uruchamiany tylko wtedy, gdy operacja przesyłania zakończy się pomyślnie lub częściowo pomyślnie.
2. **postDestinationCall** jest uruchamiany tylko wtedy, gdy operacja przesyłania zakończy się pomyślnie lub częściowo pomyślnie.
3. **SourceTransferEndExits** jest uruchamiany dla pomyślnych, częściowo zakończonych powodzeniem lub zakończonych niepowodzeniem transferów.
4. Program **postSourceCall** jest wywoływany tylko wtedy, gdy:
   - Przesyłanie nie zostało anulowane.
   - Wynik jest pomyślny lub częściowo pomyślny.
   - Wszystkie programy przesyłania po miejscu docelowym przebiegły pomyślnie.

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 .

Wyjścia użytkownika źródłowego i docelowego produktu MFT

Separatory katalogów

Separatory katalogów w specyfikacjach plików źródłowych są zawsze reprezentowane za pomocą znaków ukośnika (/), niezależnie od tego, w jaki sposób zostały określone separatory katalogów w komendzie **fteCreateTransfer** lub w IBM MQ Explorer. Należy wziąć to pod uwagę podczas pisania wyjścia. Na przykład, aby sprawdzić, czy następujący plik źródłowy istnieje: `c:\a\b.txt` i ten plik źródłowy został określony za pomocą komendy **fteCreateTransfer** lub IBM MQ Explorer, należy zauważyć, że nazwa pliku jest rzeczywiście zapisana jako: `c:/a/b.txt` Tak więc, jeśli wyszukiwany jest oryginalny łańcuch `c:\a\b.txt`, nie zostanie wyszukany zgodny.

Punkty wyjścia po stronie źródła

Przed rozpoczęciem przesyłania całego pliku

To wyjście jest wywoływane przez agenta źródłowego, gdy żądanie transferu znajduje się dalej na liście oczekujących operacji przesyłania, a przesyłanie jest zaraz rozpoczęte.

Przykładem zastosowania tego punktu wyjścia jest wysyłanie plików etapami do katalogu, do którego agent ma dostęp do odczytu/zapisu przy użyciu komendy zewnętrznej, lub do zmiany nazwy plików w systemie docelowym.

Przekaz następujące argumenty do tego wyjścia:

- Nazwa agenta źródłowego
- Nazwa agenta docelowego
- Metadane środowiska
- Metadane przesyłania
- Specyfikacje plików (w tym metadane pliku)

Dane zwracane z tego wyjścia są następujące:

- Zaktualizowane metadane przesyłania. Pozycje mogą być dodawane, modyfikowane i usuwane.
- Zaktualizowano listę specyfikacji plików, która składa się z par nazwy pliku źródłowego i nazwy pliku docelowego. Pozycje mogą być dodawane, modyfikowane i usuwane
- Indykator określający, czy przesyłanie ma być kontynuowane.
- Łańcuch, który ma zostać wstawiony do dziennika przesyłania.

Zaimplementuj interfejs SourceTransferStartExit.java interface , aby wywołać kod wyjścia użytkownika w tym punkcie wyjścia.

Po zakończeniu przesyłania całego pliku
To wyjście jest wywoływane przez agenta źródłowego po zakończeniu całego przesyłania plików.

Przykładem użycia tego punktu wyjścia jest wykonanie niektórych zadań zakończenia, takich jak wysłanie wiadomości e-mail lub wiadomość IBM MQ w celu oznaczenia, że przesyłanie zostało zakończone.

Przekaz następujące argumenty do tego wyjścia:

- Wynik wyjścia przesyłania
- Nazwa agenta źródłowego
- Nazwa agenta docelowego
- Metadane środowiska
- Metadane przesyłania
- Wyniki pliku

Dane zwracane z tego wyjścia są następujące:

- Zaktualizowano łańcuch, który ma zostać wstawiony do dziennika przesyłania.

Zaimplementuj interfejs SourceTransferEndExit.java interface , aby wywołać kod wyjścia użytkownika w tym punkcie wyjścia.

Punkty wyjścia po stronie docelowej

Przed rozpoczęciem przesyłania całego pliku
Przykładem użycia tego punktu wyjścia jest sprawdzenie poprawności uprawnień w miejscu docelowym.

Przekaz następujące argumenty do tego wyjścia:

- Nazwa agenta źródłowego
- Nazwa agenta docelowego
- Metadane środowiska
- Metadane przesyłania
- Specyfikacja pliku

Dane zwracane z tego wyjścia są następujące:

- Zaktualizowano zestaw nazw plików docelowych. Pozycje mogą być modyfikowane, ale nie dodawane ani usuwane.
- Indykator określający, czy przesyłanie ma być kontynuowane.
- Łańcuch, który ma zostać wstawiony do dziennika przesyłania.

Zaimplementuj interfejs DestinationTransferStartExit.java , aby wywołać kod wyjścia użytkownika w tym punkcie wyjścia.

Po zakończeniu przesyłania całego pliku
Przykładem użycia tego wyjścia użytkownika jest uruchomienie procesu wsadowego, w którym używane są przesłane pliki, lub wysłanie wiadomości e-mail, jeśli operacja przesyłania nie powiodła się.

Przekaz następujące argumenty do tego wyjścia:

- Wynik wyjścia przesyłania
- Nazwa agenta źródłowego
- Nazwa agenta docelowego

- Metadane środowiska
- Metadane przesyłania
- Wyniki pliku

Dane zwracane z tego wyjścia są następujące:

- Zaktualizowano łańcuch, który ma zostać wstawiony do dziennika przesyłania.

Zaimplementuj interfejs DestinationTransferEndExit.java , aby wywołać kod wyjścia użytkownika w tym punkcie wyjścia.

Pojęcia pokrewne

Interfejsy Java dla wyjść użytkownika produktu MFT

Odsyłacze pokrewne

“Włączanie debugowania zdalnego dla wyjść użytkownika produktu MFT” na stronie 1346
Podczas tworzenia programów zewnętrznych użytkownik może chcieć użyć debugera w celu ułatwienia wyszukiwania problemów w kodzie.

“Przykładowe wyjście użytkownika przesyłania źródła MFT” na stronie 1347

Wyjścia użytkownika monitora zasobów produktu MFT

Używanie wyjść użytkownika we/wy MFT

Istnieje możliwość użycia wyjść użytkownika we/wy transferu Managed File Transfer w celu skonfigurowania niestandardowego kodu w celu wykonania podstawowych operacji we/wy systemu plików dla transferów Managed File Transfer .

Zwykle w przypadku przesyłania MFT agent wybiera z jednego z wbudowanych dostawców we/wy do interakcji z odpowiednimi systemami plików dla operacji przesyłania. Wbudowane dostawcy we/wy obsługują następujące typy systemów plików:

- Zwykłe systemy plików typu UNIX-type i Windows
- z/OS z/OS sekwencyjne i partycjonowane zestawy danych (tylko w systemie z/OS)
- IBM i Rodzime zbiory składowania IBM i (tylko w systemie IBM i)
- Kolejki produktu IBM MQ
- Zdalne serwery protokołu 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 których wymagane jest niestandardowe zachowanie we/wy, można napisać wyjście użytkownika we/wy przesyłania.

Wyjścia użytkownika we/wy transferu korzystają z istniejącej infrastruktury dla wyjść użytkownika. Jednak te wyjścia użytkownika we/wy transferu różnią się od innych programów zewnętrznych, ponieważ ich funkcja jest dostępna wielokrotnie w czasie całego przesyłania dla każdego pliku.

Użyj właściwości IOExitClasses agenta (w pliku `agent.properties`), aby określić, które klasy wyjścia we/wy mają być ładowane. Każdą klasę wyjścia należy oddzielić przecinkiem, na przykład:

```
IOExitClasses=testExits.TestExit1,testExits.testExit2
```

Interfejsy Java dla wyjść użytkownika we/wy przesyłania są następujące:

Wyjście IOExit

Główny punkt wejścia używany do określenia, czy jest używane wyjście we/wy. Ta instancja jest odpowiedzialna za tworzenie instancji IOExitPath .

Dla właściwości agenta IOExitClassesnależy podać tylko interfejs wyjścia we/wy wyjścia 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 zapoznanie się ze ścieżką i ścieżkami pochodnymi do wyświetlenia. Interfejsy ścieżki IOExitResourcei ścieżki IOExitWildcardrozszerzają ścieżkę IOExitPath.

IOExitChannel
Umożliwia odczytywanie danych z zasobu IOExitPath lub ich zapisywanie do niego.

Kanał IOExitRecord
Rozszerza interfejs IOExitChannel dla zasobów IOExitPath zorientowanych na rekordy, co umożliwia odczytywanie danych z zasobu IOExitPath lub zapisywanie ich do nich w wielokrotnościach rekordów.

IOExitLock
Reprezentuje blokadę zasobu IOExitPath w celu uzyskania dostępu współużytkowanego lub wyłącznego.

z/OS **IOExitRecordResourcePath**
Rozszerza interfejs ścieżki IOExitResourcew taki sposób, aby reprezentował kontener danych dla zbioru zorientowanego na rekordy, na przykład zestawu danych z/OS . Za pomocą interfejsu można znaleźć dane i utworzyć instancje kanału IOExitRecorddla operacji odczytu lub zapisu.

Ścieżka IOExitResource
Rozszerza interfejs IOExitPath w taki sposób, aby reprezentował kontener danych, na przykład plik lub katalog. Za pomocą interfejsu można znaleźć dane. Jeśli interfejs reprezentuje katalog, można użyć metody listPaths , aby zwrócić listę ścieżek.

Ścieżka IOExitWildcard
Rozszerza interfejs IOExitPath w taki sposób, aby reprezentował ścieżkę, która oznacza znak wieloznaczny. Tego interfejsu można użyć do dopasowania wielu ścieżek IOExitResource.

IOExitProperties
Określa właściwości, które określają sposób obsługi przez program Managed File Transfer ścieżki IOExitPath dla niektórych aspektów operacji we/wy. Na przykład, czy mają być używane pliki pośrednie, czy też do ponownego odczytania zasobu od początku, jeśli operacja przesyłania jest restartowana.

IBM i Przykładowe MFT na wyjściu użytkownika IBM i

Produkt Managed File Transfer udostępnia przykładowe wyjścia użytkownika specyficzne dla produktu IBM i 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
Przykładowe pliki programu zewnętrznego com.ibm.wmqfte.exit.io.ibmi.qdls.FTEQDLSExit w systemie plików QDLS w systemie IBM isą przesyłane do systemu plików QDLS. Po zainstalowaniu wyjścia wszystkie operacje przesyłania do plików, które rozpoczynają się od produktu /QDLS , automatycznie korzystają z wyjścia.

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

Przykładowe wyjście użytkownika com.ibm.wmqfte.exit.user.ibmi.FileMemberMonitorExit jest podobne do monitora plików MFT i automatycznie przesyła elementy zbioru fizycznego z biblioteki produktu 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 ma również opcjonalne pole metadanych "naming.scheme.qsys.monitor", którego można użyć do określenia schematu nazewnictwa używanego w trakcie przesyłania. Domyślnie to pole jest ustawione na "unix", co powoduje, że plik docelowy ma być nazywany F00.MBR. Można również podać wartość "ibmi", aby użyć schematu IBM i FTP FILE.MEMBER , na przykład /QSYS.LIB/FOO.LIB/BAR.FILE/BAZ.MBR jest przesyłany jako 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 właściwości sourceTransferStartExitKlasy w pliku `agent.properties` .
3. Zrestartuj agent.

com.ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit

Przykładowe wyjście użytkownika com.ibm.wmqfte.exit.user.ibmi.EmptyFileDeleteExit usuwa pusty obiekt pliku, gdy podzbiór zbioru źródłowego zostanie usunięty jako część przesyłania. Ponieważ obiekty plików IBM i mogą potencjalnie pomieścić wiele elementów, obiekty plików są traktowane jak katalogi przez program MFT. Z tego powodu nie można wykonać operacji przenoszenia w obiekcie pliku za pomocą programu MFT; operacje przenoszenia są obsługiwane tylko na poziomie podzbioru. W związku z tym podczas wykonywania operacji przenoszenia na elemencie pozostawiany jest teraz pusty plik. Użyj tego wyjścia przykładowego, jeśli chcesz usunąć te puste pliki w ramach żądania przesyłania.

Jeśli dla metadanych "empty.file.delete" zostanie podana wartość "true" i zostanie ona przesunięta FTEFileMember, przykładowe wyjście spowoduje usunięcie pliku nadrzędnego, 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 właściwości sourceTransferStartExitKlasy w pliku `agent.properties` .
3. Zrestartuj agent.

Odsyłacze pokrewne

“Używanie wyjść użytkownika we/wy MFT” na stronie 1344
Istnieje możliwość użycia wyjść użytkownika we/wy transferu Managed File Transfer w celu skonfigurowania niestandardowego kodu w celu wykonania podstawowych operacji we/wy systemu plików dla transferów Managed File Transfer .

Właściwości agenta MFT dla programów zewnętrznych

Włączanie debugowania zdalnego dla wyjść użytkownika produktu MFT

Podczas tworzenia programów zewnętrznych użytkownik może chcieć użyć debugera w celu ułatwienia wyszukiwania problemów w kodzie.

Ponieważ wyjścia są uruchamiane na maszynie wirtualnej Java , na której działa agent, nie można używać obsługi debugowania bezpośredniego, która jest zwykle zawarta w zintegrowanym środowisku

programistycznym. Można jednak włączyć zdalne debugowanie maszyny JVM, a następnie nawiązać połączenie z odpowiednim zdalnym debugerem.

Aby włączyć debugowanie zdalne, należy użyć standardowych parametrów maszyny JVM **-Xdebug** i **-Xrunjdwp**. Te właściwości są przekazywane do maszyny JVM, na której działa agent, przez zmienną środowiskową **BFG_JVM_PROPERTIES** . Na przykład w systemie UNIX następujące komendy uruchamiają agenta i powodują, że wirtualna maszyna języka Java nasłuchuje 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 połączy się. Zamiast komendy **export** należy użyć komendy **set** w systemie Windows .

Można również użyć innych metod komunikacji między debugerem a maszyną JVM. Na przykład wirtualna maszyna języka Java może otworzyć połączenie z debugerem, a nie odwrotnie, lub użyć pamięci współużytkowanej zamiast TCP. Więcej szczegółów można znaleźć w dokumentacji Java Platform Debugger Architecture .

Jeśli agent jest uruchamiany w trybie debugowania zdalnego, 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 kompatybilne z JPDA.

1. Kliknij opcję **Uruchom** > **Otwórz okno dialogowe debugowania** (lub opcję **Uruchom** > **Konfiguracje debugowania** lub **Uruchom** > **Debuguj okno dialogowe** w zależności od używanej wersji platformy Eclipse).
2. Kliknij dwukrotnie opcję **Remote Java Application** (Zdalna aplikacja Java) na liście typów konfiguracji, aby utworzyć konfigurację debugowania.
3. Wypełnij pola konfiguracji i zapisz konfigurację debugowania. Jeśli maszyna JVM agenta została już uruchomiona w trybie debugowania, można teraz nawiązać połączenie z JVM.

Przykładowe wyjście użytkownika przesyłania źródła 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";
    }
}
```

Wyjście użytkownika referencji mostu protokołu przykładowego

Informacje na temat korzystania z tego przykładowego programu zewnętrznego można znaleźć w sekcji Odwzorowanie informacji autoryzacyjnych 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
    }
}
```

Wyjście użytkownika właściwości mostu przykładowego protokołu

Informacje na temat korzystania z tego przykładowego programu zewnętrznego można znaleźć w sekcji 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 programem MFT przez umieszczanie komunikatów w kolejce komend agenta

Użytkownik może napisać aplikację, która steruje serwerem 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:

- Utwórz przesyłanie plików
- Tworzenie zaplanowanego przesyłania plików
- Anuluj przesyłanie plików
- Anulowanie zaplanowanego przesyłania plików
- Wywołaj komendę
- Tworzenie monitora
- Usuwanie monitora
- Zwraca 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 transferu plików lub zaplanowanego przesyłania plików, wywoływania komendy lub 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 żądania monitora MFT.

`PingAgent.xsd`
Komunikaty w tym formacie mogą być używane do wysłania komendy ping do agenta w celu sprawdzenia, czy jest on aktywny. Więcej informacji na ten temat zawiera sekcja Format komunikatów żądania agenta Ping MFT.

Agent zwraca odpowiedź na komunikaty żądania. Komunikat odpowiedzi jest umieszczany w kolejce odpowiedzi, która jest zdefiniowana 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 sensowne i sterujące z innymi źródłami informacji dostępnymi w internecie oraz w przedsiębiorstwach.

Tworzenie aplikacji dla produktu MQ Telemetry przy użyciu wzorców projektowych, obrobionych przykładów, przykładowych programów, pojęć związanych z programowaniem i informacji referencyjnych.

Pojęcia pokrewne

MQ Telemetry

Przypadki użycia telemetrii

Zadania pokrewne

Instalowanie produktu MQ Telemetry

administrowanieMQ Telemetry

MQ Telemetry rozwiązywanie problemów

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 przypadku systemu IBM MQ 8.0.0 i nowszych przykładowa aplikacja kliencka nie jest już dołączona do produktu MQ Telemetry. Była to część (nie jest już dostępna) IBM Messaging Telemetry Clients SupportPac. Podobne przykładowe aplikacje są nadal swobodnie dostępne w środowisku 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:
  - `ma9c/SDK/clients/java/org.eclipse.paho.sample.mqttv3app.jar`
  - `ma9c/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 (nie jest już dostępna) IBM Messaging Telemetry Clients SupportPac, informacje na temat instalowania i uruchamiania przykładowej aplikacji znajdują się w sekcji Sprawdzanie poprawności instalacji produktu MQ Telemetry przy użyciu wiersza komend.

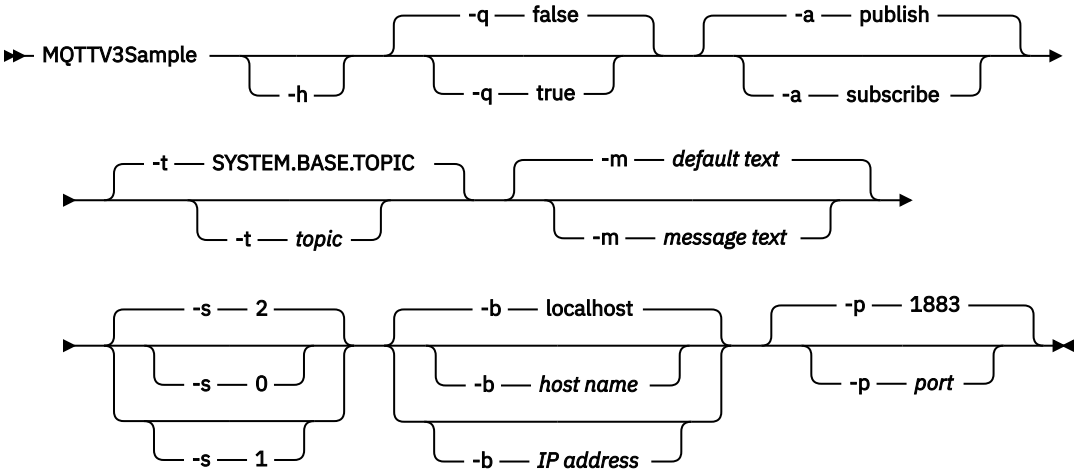
Program MQTTV3Sample

Informacje uzupełniające dotyczące przykładowej składni i parametrów dla programu `MQTTV3Sample` .

Przeznaczenie

Program `MQTTV3Sample` może być używany do publikowania komunikatów i subskrybowania tematu. Więcej informacji na temat sposobu pobierania tego programu przykładowego zawiera sekcja “IBM MQ Telemetry Transport programy przykładowe” na stronie 1354.

MQTTV3Sample syntax



Parametry

-h
Wydrukuj ten tekst pomocy i zakończ

-q
Ustaw tryb cichy, zamiast używać domyślnego trybu false.

-a
Ustaw publikowanie lub subskrybowanie, zamiast zakładać domyślne działanie publikowania.

-t
Publikowanie lub subskrybowanie tematu, zamiast publikowania lub subskrybowania tematu domyślnego

-m
Publish message text instead of sending the default publication text, "Hello from an MQTT v3 application".

-s
Ustaw wartość QoS zamiast domyślnej wartości QoS, 2.

-b
Połącz się z tą nazwą hosta lub adresem IP, a nie nawiąże połączenia z domyślną nazwą hosta localhost.

-p
Użyj tego portu, zamiast używać domyślnego, 1883.

Uruchom program MQTTV3Sample .

Aby zasubskrybować temat w systemie Windows, należy użyć komendy:

```
run MQTTV3Sample -a subscribe
```

Aby opublikować komunikat w systemie Windows, należy użyć następującej komendy:

```
run MQTTV3Sample
```

Pojęcia dotyczące programowania klienta MQTT

Pojęcia opisane w tej sekcji ułatwiają zrozumienie bibliotek klienta dla produktu MQTT protocol. Pojęcia uzupełniają dokumentację 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:
    - `ma9c/SDK/clients/java/org.eclipse.paho.sample.mqttv3app.jar`
    - `ma9c/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 odrębnego środowiska wykonawczego klienta.

Warunki licencjonowania dla klientów są powiązane z serwerem, z którym łączą się klienty.

Biblioteki klienta produktu MQTT są implementacjami referencyjnymi produktu MQTT protocol. Istnieje możliwość zaimplementowania własnych klientów w różnych językach odpowiednich dla różnych platform urządzeń. Patrz IBM MQ Telemetry Transport format i protokół.

Dokumentacja interfejsu API nie zawiera żadnych założeń dotyczących serwera MQTT , z którym klient jest połączony. Zachowanie klienta może się nieznacznie różnić w przypadku połączenia z różnymi serwerami. W poniższych opisach opisano zachowanie klienta podczas połączenia z usługą telemetryczną IBM MQ .

Procedury zwrotne i synchronizacja w aplikacjach klienckich produktu MQTT

W modelu programistycznym klienta produktu MQTT są używane obszernie wątki. Wątki dekodują aplikację kliencką MQTT , o ile tylko mogą, z opóźnień w przesyłaniu komunikatów do serwera i z serwera. Publikacje, tokeny dostarczania i zdarzenia utraconych połączeń są dostarczane do metod w klasie wywołania zwrotnego, która implementuje interfejs `MqttCallback`.

Wywołania zwrotne

Uwaga: Najnowsze zmiany w serwisie `MqttCallback`można znaleźć w serwisie WWW Eclipse Paho . Na przykład, `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)`

Program `connectionLost` jest wywoływany po wystąpieniu błędu komunikacyjnego w celu usunięcia połączenia. Jest ona również wywoływana, jeśli serwer usunie połączenie w wyniku błędu na serwerze po nawiązaniu połączenia. Błędy serwera są rejestrowane w dzienniku błędów menedżera kolejek. Serwer usuwa połączenie z klientem, a klient wywołuje `MqttCallback.connectionLost`.

Jedynymi błędami zdalnymi zgłaszanych 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, których wynikiem jest `connectionLost` , to błędy autoryzacji. Na przykład serwer telemetryczny podejmuje próbę opublikowania w temacie w imieniu klienta, który nie ma uprawnień do publikowania w tym temacie. Wszystkie wyniki w kodzie warunku `MQCC_FAIL` zwracane do serwera telemetrycznego mogą spowodować usunięcie połączenia.

`deliveryComplete(IMqttDeliveryToken token)`

Klient `deliveryComplete` jest wywoływany przez klienta MQTT w celu przekazania tokenu dostawy z powrotem do aplikacji klienckiej. Patrz “Znaczniki dostawy” na stronie 1364. 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` , dostarczenie zostanie zakończone. Do czasu zakończenia dostarczania komunikaty z funkcją `QoS` 1 lub 2 są zachowywane przez klasę trwałości.

Wywołanie funkcji `deliveryComplete` jest punktem synchronizacji między aplikacją a klasą trwałości. Metoda `deliveryComplete` nigdy nie jest wywoływana dwukrotnie dla tego samego komunikatu.

Gdy wywołanie zwrotne aplikacji zwraca wartość z `deliveryComplete` do klienta MQTT , klient wywołuje `MqttClientPersistence.remove` dla komunikatów z funkcją `QoS` 1 lub 2. Program `MqttClientPersistence.remove` usuwa lokalnie zapisaną kopię opublikowanego komunikatu.

Z perspektywy przetwarzania transakcji wywołanie funkcji `deliveryComplete` jest transakcją jednofazową, która zatwierdza dostawę. Jeśli przetwarzanie nie powiedzie się podczas wywołania zwrotnego, po ponownym uruchomieniu klienta `MqttClientPersistence.remove` zostanie ponownie wywołana w celu usunięcia lokalnej kopii opublikowanego komunikatu. Wywołanie

zwrotne nie jest ponownie wywoływane. Jeśli do przechowywania dziennika dostarczonych komunikatów używana jest procedura zwrotna, nie można zsynchronizować dziennika z klientem MQTT . Jeśli dziennik ma być niezawodnie przechowywany, należy zaktualizować dziennik w klasie `MqttClientPersistence` .

Znacznik dostawy i komunikat są przywoływane przez główny wątek aplikacji i klienta MQTT . Klient MQTT wyrejestrowuje obiekt `MqttMessage` po zakończeniu dostarczania, a także obiekt znacznika dostarczania, gdy klient rozłącza się. Obiekt `MqttMessage` może być czyszczony po zakończeniu dostarczania, jeśli aplikacja kliencka wyłuskuje go. Znacznik dostarczania może być czyszczony po rozłączeniu sesji.

Atrybuty `IMqttDeliveryToken` i `MqttMessage` można pobrać po opublikowaniu komunikatu. Jeśli po opublikowaniu komunikatu zostanie podjęta próba ustawienia atrybutów `MqttMessage` , wynik nie zostanie zdefiniowany.

Klient MQTT kontynuuje przetwarzanie potwierdzeń dostarczenia, jeśli klient ponownie łączy się z poprzednią sesją za pomocą tego samego `ClientIdentifier` ; patrz “Czyste sesje” na stronie 1360. Aplikacja kliencka MQTT musi ustawić `MqttClient.CleanSession` na `false` dla poprzedniej sesji, a następnie ustawić ją na `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 tokenów dostawy i komunikatów, wyłuskuj je. 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 klient aplikacji po zakończeniu dostarczania oczekującego. Zanim klient aplikacji połączy się, będzie mógł pobrać oczekujące dostawy za pomocą metody `MqttClient.getPendingDeliveryTokens` .

Zauważ, 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 tworzonym przez klienta. Jeśli nowa instancja klienta MQTT ponownie utworzy znacznik dostarczania, klasa `MqttClientPersistence` ponownie utworzy obiekt `MqttMessage` . For consistency `token.getMessage` returns `null` if `token.isCompleted` is `true`, regardless of whether the message object was created by the application client or the `MqttClientPersistence` class.

`messageArrived(String topic, MqttMessage message)`

Program `messageArrived` jest wywoływany po nadejściu publikacji dla klienta, który jest zgodny z tematem subskrypcji. `temat` to temat publikowania, a nie filtr subskrypcji. Dwa mogą być różne, jeśli filtr zawiera znaki wieloznaczne.

Jeśli temat pasuje do wielu subskrypcji utworzonych przez klienta, klient otrzymuje wiele kopii tej publikacji. Jeśli klient publikuje w temacie, do którego również subskrybuje, otrzymuje kopię własnej publikacji.

Jeśli komunikat jest wysyłany z usługą QoS 1 lub 2, komunikat jest zapisywany przez klasę `MqttClientPersistence` przed wywołaniami `messageArrived`klienta MQTT . `messageArrived` zachowuje się jak `deliveryComplete`: jest wywoływana tylko raz dla publikacji, a lokalna kopia publikacji jest usuwana przez program `MqttClientPersistence.remove` , gdy program `messageArrived` powróci do klienta MQTT . Klient MQTT usuwa odwołania do tematu i komunikatu, gdy program `messageArrived` powróci do klienta MQTT . Obiekty tematów i komunikatów są czyszczeniem pamięci, jeśli klient aplikacji nie ma odwołania do odwołania do obiektów.

Synchronizacja aplikacji zwrotnych, wielowątkowych i aplikacji klienckich

Klient MQTT wywołuje metodę wywołania zwrotnego w osobnym wątku do głównego wątku aplikacji. Aplikacja kliencka nie tworzy wątku dla wywołania zwrotnego, jest on tworzony przez klienta MQTT .

Klient MQTT synchronizuje metody wywołania zwrotnego. W danej chwili działa tylko jedna instancja metody wywołania zwrotnego. Synchronizacja ułatwia aktualizację obiektu, który utalentowany jest, które publikacje zostały dostarczone. Jedna instancja serwera `MqttCallback.deliveryComplete`

jest uruchamiana jednocześnie, a więc można bezpiecznie aktualizować ją bez dalszej synchronizacji. Jest to również przypadek, że tylko jedna publikacja dociera w danym momencie. Kod w metodzie `messageArrived` może aktualizować obiekt bez synchronizacji. Jeśli odwołujesz się do "tally" lub obiektu, który jest aktualizowany, w innym wątku, zsynchronizuj ten sam obiekt lub obiekt.

Znacznik dostarczania udostępnia mechanizm synchronizacji między głównym wątkiem aplikacji a publikacją. Metoda `token.waitForCompletion` oczekuje na zakończenie dostarczania konkretnej publikacji lub do momentu utraty ważności opcjonalnego limitu czasu. Produkt `token.waitForCompletion` może być używany w następujący sposób, aby można było przetworzyć jedną publikację naraz.

Aby przeprowadzić synchronizację z metodą `MqttCallback.deliveryComplete` . Tylko w przypadku, gdy program `MqttCallback.deliveryComplete` powróci do programu MQTT , zostanie wznowiony `token.waitForCompletion` . Za pomocą tego mechanizmu można synchronizować działający kod w produkcie `MqttCallback.deliveryComplete` , zanim kod zostanie uruchomiony w głównym wątku aplikacji.

Co, jeśli chcesz opublikować bez czekania na każdą publikację, która ma być dostarczona, ale chcesz potwierdzić, kiedy wszystkie publikacje zostały dostarczone? W przypadku publikowania w pojedynczym wątku ostatnia publikacja, która ma zostać wysłana, jest również ostatnią dostarczoną.

Synchronizacja żądań wysłanych do serwera

Tabela 195 na stronie 1359 opisuje metody w kliencie MQTT Java , które wysyłają żądanie do serwera. Jeśli klient aplikacji nie ustawi nieokreślonego limitu czasu, klient nigdy nie będzie czekał bezterminowo na serwer. Jeśli klient zawiesi się, jest to problem z programowaniem aplikacji lub defekt w kliencie MQTT .

Tabela 195. Zachowanie synchronizacji metod, których wynikiem są żądania do serwera

Metoda	Synchronizacja	Limit czasu
`MqttClient.Connect`	Oczekuje na nawiązanie połączenia z serwerem.	Wartość domyślna to 30 sekund lub wartość ustawiona przez parametr, a następnie zgłasza wyjątek.
`MqttClient.Disconnect`	Oczekuje na zakończenie pracy, którą musi wykonać klient MQTT , oraz rozłączenie sesji TCP/IP.	
`MqttClient.Subscribe`	Oczekuje na zakończenie działania metody Subskrybuj lub UnSubscribe .	
`MqttClient.UnSubscribe`		
`MqttClient.Publish`	Powraca natychmiast do wątku aplikacji po przekazaniu żądania do klienta MQTT .	Brak.
`IMqttDeliveryToken.waitFor Completion`	Oczekuje na zwrócenie znacznika dostarczania.	Nieokreślony lub jako parametr ustawiony jako parametr.

Pojęcia pokrewne

Czyste sesje

Klient MQTT i usługa telemetryczna (MQXR) obsługują informacje o stanie sesji. Informacje o stanie są używane do zapewnienia "co najmniej raz" i "dokładnie raz" dostarczania, a "dokładnie raz" otrzymania publikacji. Stan sesji obejmuje również subskrypcje utworzone przez klienta MQTT . Istnieje możliwość uruchomienia klienta MQTT z zachowaniem informacji o stanie między sesjami lub bez niego. Zmień tryb czyszczenia sesji, ustawiając wartość `MqttConnectOptions.cleanSession` przed nawiązaniem połączenia.

Identyfikator klienta
Identyfikator klienta to 23 bajtowy łańcuch, który identyfikuje klienta MQTT . Każdy identyfikator musi być unikalny w obrębie wszystkich połączonych w danym momencie klientów. Identyfikator musi zawierać tylko znaki poprawne w nazwie menedżera kolejek. W ramach tych ograniczeń można użyć dowolnego łańcucha identyfikacyjnego. Ważne jest, aby procedura przydzielania identyfikatorów klientów była procedurą, a także sposób konfigurowania klienta z wybranym identyfikatorem.

Znaczniki dostawy

Publikacja ostatniego testamentu i testamentu
Jeśli połączenie klienta MQTT nieoczekiwanie kończy się, można skonfigurować MQ Telemetry w taki sposób, aby wysyłał publikację "ostatni testament i testament". Należy wstępnie zdefiniować treść publikacji oraz temat, do którego ma zostać wysłana. Właściwość "last will and testament" jest właściwością połączenia. Utwórz go 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 jeden raz". Można zaimplementować własny mechanizm utrwalania na kliencie lub użyć domyślnego mechanizmu utrwalania udostępnianego wraz z klientem. Trwałość działa w obu kierunkach, w przypadku publikacji wysyłanych do klienta lub z niego.

Publikacje
Publikacje to instancje produktu `MqttMessage` , które są powiązane z łańcuchem tematu. Klienty produktu MQTT mogą tworzyć publikacje, które mają być wysyłane do produktu IBM MQ, a następnie zasubskrybować tematy w produkcie IBM MQ , aby otrzymywać publikacje.

Jakość usług świadczonych przez klienta MQTT
Klient MQTT udostępnia trzy cechy usługi umożliwiające dostarczanie publikacji do produktu IBM MQ oraz do klienta MQTT : "co najwyżej raz", "co najmniej raz" i "dokładnie jeden raz". Gdy klient MQTT wysyła żądanie do programu IBM MQ w celu utworzenia subskrypcji, żądanie jest wysyłane wraz z jakością usługi co najmniej raz.

Zachowane publikacje i klienty MQTT
Temat może mieć jedną i tylko jedną, zachowaną publikację. Jeśli subskrypcja została utworzona w temacie, który ma zachowaną publikację, publikacja jest natychmiast przekazywana do użytkownika.

Subskrypcje
Utwórz subskrypcje, aby zarejestrować zainteresowanie tematami publikacji przy użyciu filtru tematów. Klient może utworzyć wiele subskrypcji lub subskrypcję zawierającą filtr tematów, który korzysta ze znaków wieloznacznych, w celu zarejestrowania zainteresowania w wielu tematach. Publikacje dotyczące tematów zgodnych z filtrami są wysyłane do klienta. Subskrypcje mogą pozostać aktywne, gdy klient jest odłączony. Publikacje są wysyłane do klienta po ponownym nawiązaniu połączenia.

Łańcuchy tematów i filtry tematów w klientach MQTT
Łańcuch tematu i filtry tematów są używane do publikowania i subskrybowania. Składnia łańcuchów tematów i filtrów w klientach MQTT jest w dużej mierze taka sama, jak łańcuchy tematów w produkcie IBM MQ.

Czyste sesje

Klient MQTT i usługa telemetryczna (MQXR) obsługują informacje o stanie sesji. Informacje o stanie są używane do zapewnienia "co najmniej raz" i "dokładnie raz" dostarczania, a "dokładnie raz" otrzymania publikacji. Stan sesji obejmuje również subskrypcje utworzone przez klienta MQTT . Istnieje możliwość uruchomienia klienta MQTT z zachowaniem informacji o stanie między sesjami lub bez niego. Zmień tryb czyszczenia sesji, ustawiając wartość `MqttConnectOptions.cleanSession` przed nawiązaniem połączenia.

Podczas łączenia aplikacji klienckiej MQTT za pomocą metody `MqttClient.connect` klient identyfikuje połączenie przy użyciu identyfikatora klienta i adresu serwera. Serwer sprawdza, czy informacje o sesji zostały zeskładowane z poprzedniego połączenia z serwerem. Jeśli poprzednia sesja nadal istnieje, a następnie `cleanSession=true`, to poprzednie informacje o sesji na kliencie i serwerze zostaną wyczyszczone. Jeśli `cleanSession=false` poprzednia sesja została wznowiona. Jeśli żadna wcześniejsza sesja nie istnieje, zostanie uruchomiona nowa sesja.

Uwaga: Administrator produktu IBM MQ może wymusić zamknięcie otwartej sesji i usunąć wszystkie informacje o sesji. Jeśli klient ponownie otworzy sesję z programem `cleanSession=false`, uruchamiana jest nowa sesja.

Publikacje

Jeśli przed nawiązaniem połączenia z klientem zostanie użyta wartość domyślna `MqttConnectOptions`lub zostanie ustawiona wartość `MqttConnectOptions.cleanSession` na `true` , wszystkie oczekujące dostawy publikacji dla klienta zostaną usunięte po nawiązaniu połączenia przez klienta.

Ustawienie sesji czyszczenia nie ma wpływu na publikacje wysłane z produktem `QoS=0`. W przypadku produktów `QoS=1` i `QoS=2`użycie produktu `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

Procedury zwrotne i synchronizacja w aplikacjach klienckich produktu MQTT
W modelu programistycznym klienta produktu MQTT są używane obszernie wątki. Wątki dekodują aplikację kliencką MQTT , o ile tylko mogą, z opóźnień w przesyłaniu komunikatów do serwera i z serwera. Publikacje, tokeny dostarczania i zdarzenia utraconych połączeń są dostarczane do metod w klasie wywołania zwrotnego, która implementuje interfejs `MqttCallback`.

Identyfikator klienta
Identyfikator klienta to 23 bajtowy łańcuch, który identyfikuje klienta MQTT . Każdy identyfikator musi być unikalny w obrębie wszystkich połączonych w danym momencie klientów. Identyfikator musi zawierać tylko znaki poprawne w nazwie menedżera kolejek. W ramach tych ograniczeń można użyć dowolnego łańcucha identyfikacyjnego. Ważne jest, aby procedura przydzielania identyfikatorów klientów była procedurą, a także sposób konfigurowania klienta z wybranym identyfikatorem.

Znaczniki dostawy

Publikacja ostatniego testamentu i testamentu
Jeśli połączenie klienta MQTT nieoczekiwanie kończy się, można skonfigurować MQ Telemetry w taki sposób, aby wysyłał publikację "ostatni testament i testament". Należy wstępnie zdefiniować treść publikacji oraz temat, do którego ma zostać wysłana. Właściwość "last will and testament" jest właściwością połączenia. Utwórz go 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 jeden raz". Można zaimplementować własny mechanizm utrwalania na kliencie lub użyć domyślnego mechanizmu utrwalania udostępnianego wraz z klientem. Trwałość działa w obu kierunkach, w przypadku publikacji wysyłanych do klienta lub z niego.

Publikacje
Publikacje to instancje produktu `MqttMessage` , które są powiązane z łańcuchem tematu. Klienty produktu MQTT mogą tworzyć publikacje, które mają być wysyłane do produktu IBM MQ, a następnie zasubskrybować tematy w produkcie IBM MQ , aby otrzymywać publikacje.

Jakość usług świadczonych przez klienta MQTT
Klient MQTT udostępnia trzy cechy usługi umożliwiające dostarczanie publikacji do produktu IBM MQ oraz do klienta MQTT : "co najwyżej raz", "co najmniej raz" i "dokładnie jeden raz". Gdy klient MQTT wysyła żądanie do programu IBM MQ w celu utworzenia subskrypcji, żądanie jest wysyłane wraz z jakością usługi co najmniej raz.

Zachowane publikacje i klienty MQTT
Temat może mieć jedną i tylko jedną, zachowaną publikację. Jeśli subskrypcja została utworzona w temacie, który ma zachowaną publikację, publikacja jest natychmiast przekazywana do użytkownika.

Subskrypcje
Utwórz subskrypcje, aby zarejestrować zainteresowanie tematami publikacji przy użyciu filtru tematów. Klient może utworzyć wiele subskrypcji lub subskrypcję zawierającą filtr tematów, który korzysta ze znaków wieloznacznych, w celu zarejestrowania zainteresowania w wielu tematach. Publikacje dotyczące tematów zgodnych z filtrami są wysyłane do klienta. Subskrypcje mogą pozostać aktywne, gdy klient jest odłączony. Publikacje są wysyłane do klienta po ponownym nawiązaniu połączenia.

Łańcuchy tematów i filtry tematów w klientach MQTT
Łańcuch tematu i filtry tematów są używane do publikowania i subskrybowania. Składnia łańcuchów tematów i filtrów w klientach MQTT jest w dużej mierze taka sama, jak łańcuchy tematów w produkcie IBM MQ.

Identyfikator klienta

Identyfikator klienta to 23 bajtowy łańcuch, który identyfikuje klienta MQTT . Każdy identyfikator musi być unikalny w obrębie wszystkich połączonych w danym momencie klientów. Identyfikator musi zawierać tylko znaki poprawne w nazwie menedżera kolejek. W ramach tych ograniczeń można użyć dowolnego łańcucha identyfikacyjnego. Ważne jest, aby procedura przydzielania identyfikatorów klientów była procedurą, a także sposób konfigurowania klienta z wybranym identyfikatorem.

Identyfikator klienta jest używany w administrowaniu systemem MQTT . Z potencjalnie setkami tysięcy klientów do administrowania, konieczne jest szybkie zidentyfikowanie konkretnego klienta. Załóżmy na przykład, że urządzenie nie działa, a użytkownik jest powiadamiany, być może przez klienta dzwoniący 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 za pomocą połączeń klienckich MQTT każde połączenie jest oznaczone identyfikatorem klienta. Aby podjąć decyzję o tym, jak najlepiej odwzorować ten identyfikator na urządzenie i serwer, należy zadać sobie następujące pytania:

- Czy wygodnie jest utrzymywać i korzystać z bazy danych, która odwzorowuje każde urządzenie na identyfikator klienta i na serwer?
- Czy nazwa urządzenia może zidentyfikować serwer, do którego jest przyłączony?
- Czy potrzebna jest tabela wyszukiwania, która odwzorowuje identyfikator klienta na urządzenie fizyczne?
- Czy identyfikator klienta identyfikuje określone urządzenie, użytkownika lub aplikację działającą na kliencie?
- Jeśli klient zastąpi wadliwe urządzenie nowym urządzeniem, to czy nowe urządzenie ma ten sam identyfikator co stare urządzenie, czy też jest przydzielany nowy identyfikator? (Jeśli zmienisz urządzenie fizyczne i zachowaj ten sam identyfikator, oczekujące publikacje i aktywne subskrypcje zostaną automatycznie przesłane do nowego urządzenia).

Potrzebny jest także system zapewniający unikalność identyfikatorów klientów, a użytkownik musi mieć niezawodny proces ustawiania identyfikatora na kliencie. Jeśli urządzeniem klienckim jest "black-box", bez interfejsu użytkownika, można było wyprodukować urządzenie z identyfikatorem klienta lub być może proces instalacji i konfiguracji oprogramowania, który skonfiguruje urządzenie przed jego aktywowaniem.

Aby identyfikator był krótki i unikalny, można utworzyć identyfikator klienta na podstawie adresu MAC urządzenia 48 bitowego. Jeśli wielkość transmisji nie jest problemem krytycznym, można użyć pozostałych 17 bajtów, aby ułatwić administrowanie adresem.

Pojęcia pokrewne

Procedury zwrotne i synchronizacja w aplikacjach klienckich produktu MQTT
W modelu programistycznym klienta produktu MQTT są używane obszernie wątki. Wątki dekodują aplikację kliencką MQTT , o ile tylko mogą, z opóźnień w przesyłaniu komunikatów do serwera i z serwera. Publikacje, tokeny dostarczania i zdarzenia utraconych połączeń są dostarczane do metod w klasie wywołania zwrotnego, która implementuje interfejs `MqttCallback`.

Czyste sesje
Klient MQTT i usługa telemetryczna (MQXR) obsługują informacje o stanie sesji. Informacje o stanie są używane do zapewnienia "co najmniej raz" i "dokładnie raz" dostarczania, a "dokładnie raz" otrzymania publikacji. Stan sesji obejmuje również subskrypcje utworzone przez klienta MQTT . Istnieje możliwość uruchomienia klienta MQTT z zachowaniem informacji o stanie między sesjami lub bez niego. Zmień tryb czyszczenia sesji, ustawiając wartość `MqttConnectOptions.cleanSession` przed nawiązaniem połączenia.

Znaczniki dostawy

Publikacja ostatniego testamentu i testamentu
Jeśli połączenie klienta MQTT nieoczekiwanie kończy się, można skonfigurować MQ Telemetry w taki sposób, aby wysyłał publikację "ostatni testament i testament". Należy wstępnie zdefiniować treść publikacji oraz temat, do którego ma zostać wysłana. Właściwość "last will and testament" jest właściwością połączenia. Utwórz go 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 jeden raz". Można zaimplementować własny mechanizm utrwalania na kliencie lub użyć domyślnego mechanizmu utrwalania udostępnianego wraz z klientem. Trwałość działa w obu kierunkach, w przypadku publikacji wysyłanych do klienta lub z niego.

Publikacje
Publikacje to instancje produktu `MqttMessage` , które są powiązane z łańcuchem tematu. Klienty produktu MQTT mogą tworzyć publikacje, które mają być wysyłane do produktu IBM MQ, a następnie zasubskrybować tematy w produkcie IBM MQ , aby otrzymywać publikacje.

Jakość usług świadczonych przez klienta MQTT
Klient MQTT udostępnia trzy cechy usługi umożliwiające dostarczanie publikacji do produktu IBM MQ oraz do klienta MQTT : "co najwyżej raz", "co najmniej raz" i "dokładnie jeden raz". Gdy klient MQTT wysyła żądanie do programu IBM MQ w celu utworzenia subskrypcji, żądanie jest wysyłane wraz z jakością usługi co najmniej raz.

Zachowane publikacje i klienty MQTT
Temat może mieć jedną i tylko jedną, zachowaną publikację. Jeśli subskrypcja została utworzona w temacie, który ma zachowaną publikację, publikacja jest natychmiast przekazywana do użytkownika.

Subskrypcje
Utwórz subskrypcje, aby zarejestrować zainteresowanie tematami publikacji przy użyciu filtru tematów. Klient może utworzyć wiele subskrypcji lub subskrypcję zawierającą filtr tematów, który korzysta ze znaków wieloznacznych, w celu zarejestrowania zainteresowania w wielu tematach. Publikacje dotyczące tematów zgodnych z filtrami są wysyłane do klienta. Subskrypcje mogą pozostać aktywne, gdy klient jest odłączony. Publikacje są wysyłane do klienta po ponownym nawiązaniu połączenia.

Łańcuchy tematów i filtry tematów w klientach MQTT
Łańcuch tematu i filtry tematów są używane do publikowania i subskrybowania. Składnia łańcuchów tematów i filtrów w klientach MQTT jest w dużej mierze taka sama, jak łańcuchy tematów w produkcie IBM MQ.

Znaczniki dostawy

Gdy klient publikuje w temacie nowy znacznik dostarczania, zostanie utworzony. Za pomocą znacznika dostarczania można monitorować dostarczanie publikacji lub blokować aplikację kliencką do czasu zakończenia dostawy.

Token jest obiektem `MqttDeliveryToken` . Jest on tworzony przez wywołanie metody MqttTopic.publish () i jest zachowywany przez klienta MQTT do momentu, gdy sesja klienta zostanie rozłączona i dostarczenie zostanie zakończone.

Normalny sposób użycia tokenu polega na sprawdzeniu, czy dostawa jest kompletna. Zablokuj aplikację kliencką do czasu zakończenia dostarczania za pomocą zwróconego znacznika do wywołania `token.waitForCompletion`. Alternatywnie można podać procedurę obsługi produktu `MqttCallBack` . Po otrzymaniu przez klienta MQTT wszystkich potwierdzeń, których oczekuje w ramach dostarczania publikacji, wywołuje on `MqttCallBack.deliveryComplete` przekazując znacznik dostarczania jako parametr.

Dopóki dostawa nie zostanie zakończona, można sprawdzić publikację za pomocą zwróconego znacznika dostarczania, wywołując komendę `token.getMessage`.

Zrealizowane dostawy

Zakończenie dostaw jest asynchroniczne i zależy od jakości usługi powiązanej z publikacją.

Najwyżej raz

`QoS=0`

Dostarczanie jest kompletne natychmiast po powrocie z produktu `MqttTopic.publish`. `MqttCallback.deliveryComplete` jest wywoływana natychmiast.

Co najmniej raz

`QoS=1`

Dostarczanie jest kompletne, gdy potwierdzenie publikacji zostało odebrane z menedżera kolejek. `MqttCallback.deliveryComplete` jest wywoływana po odebraniu potwierdzenia. Komunikat może zostać dostarczony więcej niż jeden raz przed wywołaniem programu `MqttCallback.deliveryComplete` , jeśli komunikacja jest powolna lub niewiarygodna.

Dokładnie jeden raz

`QoS=2`

Dostarczanie jest kompletne, gdy klient otrzymuje komunikat o zakończeniu, że publikacja została opublikowana w subskrybentach. `MqttCallback.deliveryComplete` jest wywoływana natychmiast po odebraniu komunikatu publikacji. Nie czeka na komunikat o zakończeniu.

W rzadkich przypadkach aplikacja kliencka może nie wrócić do klienta MQTT z `MqttCallback.deliveryComplete` normalnie. Wiadomo, że dostarczenie zostało zakończone, ponieważ wywołano program `MqttCallback.deliveryComplete` . Jeśli klient zrestartuje tę samą sesję, program `MqttCallback.deliveryComplete` nie zostanie wywołany ponownie.

Dostawy niekompletne

Jeśli dostawa nie zostanie zakończona po rozłączeniu sesji klienta, można ponownie nawiązać połączenie z klientem i zakończyć dostarczanie. Dostarczanie 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 nawiąże połączenie, ustawiając ponownie atrybut `cleanSession` `MqttConnectionOptions` na wartość `false` . Jeśli parametr `cleanSession` zostanie ustawiony na wartość `true`, oczekujące tokeny dostarczania zostaną odrzucone.

Istnieje możliwość sprawdzenia, czy istnieją oczekujące dostawy, wywołując komendę `MqttClient.getPendingDeliveryTokens`. Program `MqttClient.getPendingDeliveryTokens` można wywołać przed nawiązaniem połączenia z klientem.

Pojęcia pokrewne

Procedury zwrotne i synchronizacja w aplikacjach klienckich produktu MQTT
W modelu programistycznym klienta produktu MQTT są używane obszernie wątki. Wątki dekodują aplikację kliencką MQTT , o ile tylko mogą, z opóźnień w przesyłaniu komunikatów do serwera i z serwera. Publikacje, tokeny dostarczania i zdarzenia utraconych połączeń są dostarczane do metod w klasie wywołania zwrotnego, która implementuje interfejs `MqttCallback`.

Czyste sesje
Klient MQTT i usługa telemetryczna (MQXR) obsługują informacje o stanie sesji. Informacje o stanie są używane do zapewnienia "co najmniej raz" i "dokładnie raz" dostarczania, a "dokładnie raz" otrzymania publikacji. Stan sesji obejmuje również subskrypcje utworzone przez klienta MQTT . Istnieje możliwość uruchomienia klienta MQTT z zachowaniem informacji o stanie między sesjami lub bez niego. Zmień tryb czyszczenia sesji, ustawiając wartość `MqttConnectOptions.cleanSession` przed nawiązaniem połączenia.

Identyfikator klienta
Identyfikator klienta to 23 bajtowy łańcuch, który identyfikuje klienta MQTT . Każdy identyfikator musi być unikalny w obrębie wszystkich połączonych w danym momencie klientów. Identyfikator musi zawierać tylko znaki poprawne w nazwie menedżera kolejek. W ramach tych ograniczeń można użyć dowolnego łańcucha identyfikacyjnego. Ważne jest, aby procedura przydzielania identyfikatorów klientów była procedurą, a także sposób konfigurowania klienta z wybranym identyfikatorem.

Publikacja ostatniego testamentu i testamentu
Jeśli połączenie klienta MQTT nieoczekiwanie kończy się, można skonfigurować MQ Telemetry w taki sposób, aby wysyłał publikację "ostatni testament i testament". Należy wstępnie zdefiniować treść publikacji oraz temat, do którego ma zostać wysłana. Właściwość "last will and testament" jest właściwością połączenia. Utwórz go 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 jeden raz". Można zaimplementować własny mechanizm utrwalania na kliencie lub użyć domyślnego mechanizmu utrwalania udostępnianego wraz z klientem. Trwałość działa w obu kierunkach, w przypadku publikacji wysyłanych do klienta lub z niego.

Publikacje
Publikacje to instancje produktu `MqttMessage` , które są powiązane z łańcuchem tematu. Klienty produktu MQTT mogą tworzyć publikacje, które mają być wysyłane do produktu IBM MQ, a następnie zasubskrybować tematy w produkcie IBM MQ , aby otrzymywać publikacje.

Jakość usług świadczonych przez klienta MQTT
Klient MQTT udostępnia trzy cechy usługi umożliwiające dostarczanie publikacji do produktu IBM MQ oraz do klienta MQTT : "co najwyżej raz", "co najmniej raz" i "dokładnie jeden raz". Gdy klient MQTT wysyła żądanie do programu IBM MQ w celu utworzenia subskrypcji, żądanie jest wysyłane wraz z jakością usługi co najmniej raz.

Zachowane publikacje i klienty MQTT
Temat może mieć jedną i tylko jedną, zachowaną publikację. Jeśli subskrypcja została utworzona w temacie, który ma zachowaną publikację, publikacja jest natychmiast przekazywana do użytkownika.

Subskrypcje
Utwórz subskrypcje, aby zarejestrować zainteresowanie tematami publikacji przy użyciu filtru tematów. Klient może utworzyć wiele subskrypcji lub subskrypcję zawierającą filtr tematów, który korzysta ze znaków wieloznacznych, w celu zarejestrowania zainteresowania w wielu tematach. Publikacje dotyczące tematów zgodnych z filtrami są wysyłane do klienta. Subskrypcje mogą pozostać aktywne, gdy klient jest odłączony. Publikacje są wysyłane do klienta po ponownym nawiązaniu połączenia.

Łańcuchy tematów i filtry tematów w klientach MQTT

Łańcuch tematu i filtry tematów są używane do publikowania i subskrybowania. Składnia łańcuchów tematów i filtrów w klientach MQTT jest w dużej mierze taka sama, jak łańcuchy tematów w produkcie IBM MQ.

Publikacja ostatniego testamentu i testamentu

Jeśli połączenie klienta MQTT nieoczekiwanie kończy się, można skonfigurować MQ Telemetry w taki sposób, aby wysyłał publikację "ostatni testament i testament". Należy wstępnie zdefiniować treść publikacji oraz temat, do którego ma zostać wysłana. Właściwość "last will and testament" jest właściwością połączenia. Utwórz go przed nawiązaniem połączenia z klientem.

Utwórz temat dla ostatniej testamentu i testamentu. Użytkownik może utworzyć temat, taki jak `MQTTManagement/Connections/`*`server URI`*`/`*`client identifer`*`/Lost`.

Skonfiguruj "ostatni testament i testament" przy użyciu metody `MqttConnectionOptions.setWill(MqttTopic lastWillTopic, byte [] lastWillPayload, int lastWillQos, boolean lastWillRetained)`.

Rozważ utworzenie znacznika czasu w komunikacie `lastWillPayload` . Dołączanie innych informacji o kliencie, które pomagają w identyfikacji klienta i okolicznościach połączenia. Przekaz obiekt `MqttConnectionOptions` do konstruktora `MqttClient` .

Set `lastWillQos` to 1 or 2, to make the message persistent in IBM MQ, and to guarantee delivery. Aby zachować ostatnie utracone informacje o połączeniu, ustaw wartość `lastWillRetained` na `true`.

Publikacja "last will and testament" jest wysyłana do subskrybentów, jeśli połączenie zostanie nieoczekiwanie zakończone. Jest on wysyłany, jeśli połączenie kończy się bez wywołania metody `MqttClient.disconnect` przez klienta.

Aby monitorować połączenia, należy uzupełnić publikację "ostatni testament i testament" z innymi publikacjami, aby rejestrować połączenia i programowane rozłączenia.

Pojęcia pokrewne

Procedury zwrotne i synchronizacja w aplikacjach klienckich produktu MQTT
W modelu programistycznym klienta produktu MQTT są używane obszernie wątki. Wątki dekodują aplikację kliencką MQTT , o ile tylko mogą, z opóźnień w przesyłaniu komunikatów do serwera i z serwera. Publikacje, tokeny dostarczania i zdarzenia utraconych połączeń są dostarczane do metod w klasie wywołania zwrotnego, która implementuje interfejs `MqttCallback`.

Czyste sesje
Klient MQTT i usługa telemetryczna (MQXR) obsługują informacje o stanie sesji. Informacje o stanie są używane do zapewnienia "co najmniej raz" i "dokładnie raz" dostarczania, a "dokładnie raz" otrzymania publikacji. Stan sesji obejmuje również subskrypcje utworzone przez klienta MQTT . Istnieje możliwość uruchomienia klienta MQTT z zachowaniem informacji o stanie między sesjami lub bez niego. Zmień tryb czyszczenia sesji, ustawiając wartość `MqttConnectOptions.cleanSession` przed nawiązaniem połączenia.

Identyfikator klienta
Identyfikator klienta to 23 bajtowy łańcuch, który identyfikuje klienta MQTT . Każdy identyfikator musi być unikalny w obrębie wszystkich połączonych w danym momencie klientów. Identyfikator musi zawierać tylko znaki poprawne w nazwie menedżera kolejek. W ramach tych ograniczeń można użyć dowolnego łańcucha identyfikacyjnego. Ważne jest, aby procedura przydzielania identyfikatorów klientów była procedurą, a także sposób konfigurowania klienta z wybranym identyfikatorem.

Znaczniki dostawy

Trwałość komunikatu w klientach MQTT
Komunikaty publikacji są trwałe, jeśli są wysyłane z jakością usługi "co najmniej raz"lub "dokładnie jeden raz". Można zaimplementować własny mechanizm utrwalania na kliencie lub użyć domyślnego mechanizmu utrwalania udostępnianego wraz z klientem. Trwałość działa w obu kierunkach, w przypadku publikacji wysyłanych do klienta lub z niego.

Publikacje

Publikacje to instancje produktu `MqttMessage` , które są powiązane z łańcuchem tematu. Klienty produktu MQTT mogą tworzyć publikacje, które mają być wysyłane do produktu IBM MQ, a następnie zasubskrybować tematy w produkcie IBM MQ , aby otrzymywać publikacje.

Jakość usług świadczonych przez klienta MQTT
Klient MQTT udostępnia trzy cechy usługi umożliwiające dostarczanie publikacji do produktu IBM MQ oraz do klienta MQTT : "co najwyżej raz", "co najmniej raz" i "dokładnie jeden raz". Gdy klient MQTT wysyła żądanie do programu IBM MQ w celu utworzenia subskrypcji, żądanie jest wysyłane wraz z jakością usługi co najmniej raz.

Zachowane publikacje i klienty MQTT
Temat może mieć jedną i tylko jedną, zachowaną publikację. Jeśli subskrypcja została utworzona w temacie, który ma zachowaną publikację, publikacja jest natychmiast przekazywana do użytkownika.

Subskrypcje
Utwórz subskrypcje, aby zarejestrować zainteresowanie tematami publikacji przy użyciu filtru tematów. Klient może utworzyć wiele subskrypcji lub subskrypcję zawierającą filtr tematów, który korzysta ze znaków wieloznacznych, w celu zarejestrowania zainteresowania w wielu tematach. Publikacje dotyczące tematów zgodnych z filtrami są wysyłane do klienta. Subskrypcje mogą pozostać aktywne, gdy klient jest odłączony. Publikacje są wysyłane do klienta po ponownym nawiązaniu połączenia.

Łańcuchy tematów i filtry tematów w klientach MQTT
Łańcuch tematu i filtry tematów są używane do publikowania i subskrybowania. Składnia łańcuchów tematów i filtrów w klientach MQTT jest w dużej mierze taka sama, jak łańcuchy tematów w produkcie IBM MQ.

Trwałość komunikatu w klientach MQTT

Komunikaty publikacji są trwałe, jeśli są wysyłane z jakością usługi "co najmniej raz"lub "dokładnie jeden raz". Można zaimplementować własny mechanizm utrwalania na kliencie lub użyć domyślnego mechanizmu utrwalania udostępnianego wraz z klientem. Trwałość działa w obu kierunkach, w przypadku publikacji wysyłanych do klienta lub z niego.

W produkcie MQTTtrwałość komunikatu ma dwa aspekty: sposób przesyłania komunikatu oraz informację o tym, czy jest on umieszczany w kolejce w produkcie IBM MessageSight , jak i w produkcie IBM MQ jako komunikat trwały.

1. Trwałość komunikatu dla par klienta MQTT z jakością usługi. W zależności od tego, jaką jakość usługi wybierzesz dla komunikatu, komunikat jest trwały. Trwałość komunikatu jest niezbędna do zaimplementowania wymaganej jakości usługi.

   Jeśli zostanie określona wartość "at most once"(najwyżej raz), `QoS=0`(at most once), klient odrzuci komunikat zaraz po jego opublikowaniu. Jeśli w procesie przetwarzania komunikatu wystąpi awaria, komunikat nie zostanie wysłany ponownie. Nawet jeśli klient nadal będzie aktywny, komunikat nie zostanie wysłany ponownie. Zachowanie komunikatów produktu `QoS=0` jest takie samo, jak szybkie komunikaty nietrwałe produktu IBM MQ .

   Jeśli komunikat jest publikowany przez klienta z funkcją `QoS` 1 lub 2, jest on trwały. Komunikat jest przechowywany lokalnie i usuwany z klienta tylko wtedy, gdy nie jest już potrzebny do zagwarantowania "co najmniej raz", `QoS=1`lub "dokładnie raz", `QoS=2`, dostarczania.
2. Jeśli komunikat jest oznaczony jako `QoS` 1 lub 2, jest on umieszczany w kolejce w IBM MessageSight i IBM MQ jako komunikat trwały. Jeśli jest ona oznaczona jako `QoS=0`, to jest umieszczana w kolejce IBM MessageSight i IBM MQ jako komunikat nietrwały. W produkcie IBM MQ komunikaty nietrwałe są przesyłane między menedżerami kolejek "dokładnie jeden raz", chyba że dla kanału komunikatów atrybut `NPMSPEED` jest ustawiony na wartość `FAST`.

Trwała publikacja jest przechowywana na kliencie, dopóki nie zostanie odebrana przez aplikację kliencką. W przypadku systemu `QoS=2`publikacja jest odrzucana od klienta, gdy wywołanie zwrotne aplikacji zwraca element sterujący. W przypadku `QoS=1` aplikacja może otrzymać publikację ponownie, jeśli wystąpi awaria. W przypadku systemu `QoS=0`wywołanie zwrotne odbiera publikację nie więcej niż raz. Może ona nie otrzymać publikacji, jeśli wystąpi awaria lub jeśli klient został odłączony w momencie publikacji.

Po zasubskrybowaniu tematu można zmniejszyć QoS , za pomocą którego subskrybent odbiera komunikaty, aby dopasować jego możliwości trwałości. Publikacje utworzone w wyższej jakości QoS są wysyłane z najwyższą jakością QoS , o którą zażądał subskrybent.

Zapisywanie komunikatów

Implementacja przechowywania danych na małych urządzeniach zmienia się bardzo wiele. Model tymczasowego zapisywania trwałych komunikatów w pamięci masowej, który jest zarządzany przez klienta MQTT , może być zbyt wolny lub wymagać zbyt dużej ilości pamięci masowej. W urządzeniach przenośnych mobilny system operacyjny może udostępniać usługę pamięci masowej, która jest idealna dla komunikatów produktu MQTT .

Aby zapewnić elastyczność w spełnianiu ograniczeń dotyczących małych urządzeń, klient MQTT ma dwa interfejsy trwałości. Interfejsy definiują operacje, które są związane z zapisywaniem 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 odpowiadał urządzeniu. Klient MQTT , który działa w systemie Java SE, ma domyślną implementację interfejsów, które przechowują trwałe komunikaty w systemie plików. Używa pakietu `java.io` .

Klasy trwałości

`MqttClientPersistence`
Przekaz instancji implementacji produktu `MqttClientPersistence` do klienta MQTT jako parametr konstruktora `MqttClient` . Jeśli parametr `MqttClientPersistence` zostanie pominięty z konstruktora `MqttClient` , klient MQTT zapisuje komunikaty trwałe przy użyciu klasy `MqttDefaultFilePersistence`.

`MqttPersistable`
Program `MqttClientPersistence` pobiera i umieszcza obiekty `MqttPersistable` przy użyciu klucza pamięci masowej. Jeśli nie jest używana `MqttDefaultFilePersistence`, należy udostępnić implementację produktu `MqttPersistable` oraz implementację produktu `MqttClientPersistence` .

`MqttDefaultFilePersistence`

Klient MQTT udostępnia klasę `MqttDefaultFilePersistence` . W przypadku tworzenia instancji produktu `MqttDefaultFilePersistence` w aplikacji klienckiej można udostępnić katalog, w którym będą przechowywane komunikaty trwałe jako parametr konstruktora `MqttDefaultFilePersistence` .

Alternatywnie klient MQTT może utworzyć instancję produktu `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, to ścieżka jest wartością właściwości systemowej `usr.data`, gdzie

- `rcp.data` to właściwość powiązana z instalacją platformy OSGi lub Eclipse Rich Client Platform (RCP).
- `usr.data` to katalog, w którym uruchomiono komendę Java , która uruchomiła aplikację.

Pojęcia pokrewne
Procedury zwrotne i synchronizacja w aplikacjach klienckich produktu MQTT
W modelu programistycznym klienta produktu MQTT są używane obszernie wątki. Wątki dekodują aplikację kliencką MQTT , o ile tylko mogą, z opóźnień w przesyłaniu komunikatów do serwera i z serwera.

Publikacje, tokeny dostarczania i zdarzenia utraconych połączeń są dostarczane do metod w klasie wywołania zwrotnego, która implementuje interfejs `MqttCallback`.

Czyste sesje
Klient MQTT i usługa telemetryczna (MQXR) obsługują informacje o stanie sesji. Informacje o stanie są używane do zapewnienia "co najmniej raz" i "dokładnie raz" dostarczania, a "dokładnie raz" otrzymania publikacji. Stan sesji obejmuje również subskrypcje utworzone przez klienta MQTT . Istnieje możliwość uruchomienia klienta MQTT z zachowaniem informacji o stanie między sesjami lub bez niego. Zmień tryb czyszczenia sesji, ustawiając wartość `MqttConnectOptions.cleanSession` przed nawiązaniem połączenia.

Identyfikator klienta
Identyfikator klienta to 23 bajtowy łańcuch, który identyfikuje klienta MQTT . Każdy identyfikator musi być unikalny w obrębie wszystkich połączonych w danym momencie klientów. Identyfikator musi zawierać tylko znaki poprawne w nazwie menedżera kolejek. W ramach tych ograniczeń można użyć dowolnego łańcucha identyfikacyjnego. Ważne jest, aby procedura przydzielania identyfikatorów klientów była procedurą, a także sposób konfigurowania klienta z wybranym identyfikatorem.

Znaczniki dostawy

Publikacja ostatniego testamentu i testamentu
Jeśli połączenie klienta MQTT nieoczekiwanie kończy się, można skonfigurować MQ Telemetry w taki sposób, aby wysyłał publikację "ostatni testament i testament". Należy wstępnie zdefiniować treść publikacji oraz temat, do którego ma zostać wysłana. Właściwość "last will and testament" jest właściwością połączenia. Utwórz go przed nawiązaniem połączenia z klientem.

Publikacje
Publikacje to instancje produktu `MqttMessage` , które są powiązane z łańcuchem tematu. Klienty produktu MQTT mogą tworzyć publikacje, które mają być wysyłane do produktu IBM MQ, a następnie zasubskrybować tematy w produkcie IBM MQ , aby otrzymywać publikacje.

Jakość usług świadczonych przez klienta MQTT
Klient MQTT udostępnia trzy cechy usługi umożliwiające dostarczanie publikacji do produktu IBM MQ oraz do klienta MQTT : "co najwyżej raz", "co najmniej raz" i "dokładnie jeden raz". Gdy klient MQTT wysyła żądanie do programu IBM MQ w celu utworzenia subskrypcji, żądanie jest wysyłane wraz z jakością usługi co najmniej raz.

Zachowane publikacje i klienty MQTT
Temat może mieć jedną i tylko jedną, zachowaną publikację. Jeśli subskrypcja została utworzona w temacie, który ma zachowaną publikację, publikacja jest natychmiast przekazywana do użytkownika.

Subskrypcje
Utwórz subskrypcje, aby zarejestrować zainteresowanie tematami publikacji przy użyciu filtru tematów. Klient może utworzyć wiele subskrypcji lub subskrypcję zawierającą filtr tematów, który korzysta ze znaków wieloznacznych, w celu zarejestrowania zainteresowania w wielu tematach. Publikacje dotyczące tematów zgodnych z filtrami są wysyłane do klienta. Subskrypcje mogą pozostać aktywne, gdy klient jest odłączony. Publikacje są wysyłane do klienta po ponownym nawiązaniu połączenia.

Łańcuchy tematów i filtry tematów w klientach MQTT
Łańcuch tematu i filtry tematów są używane do publikowania i subskrybowania. Składnia łańcuchów tematów i filtrów w klientach MQTT jest w dużej mierze taka sama, jak łańcuchy tematów w produkcie IBM MQ.

Publikacje

Publikacje to instancje produktu `MqttMessage` , które są powiązane z łańcuchem tematu. Klienty produktu MQTT mogą tworzyć publikacje, które mają być wysyłane do produktu IBM MQ, a następnie zasubskrybować tematy w produkcie IBM MQ , aby otrzymywać publikacje.

`MqttMessage` ma tablicę bajtów jako jej ładunek. Dążą do tego, aby wiadomości były jak najmniejsze. Maksymalna długość komunikatu dozwolona przez MQTT protocol wynosi 250 MB.

Zwykle program kliencki MQTT używa produktu `java.lang.String` lub `java.lang.StringBuffer` do manipulowania treścią komunikatu. Dla wygody klasa `MqttMessage` ma metodę `toString` , która

umożliwia przekształcenie jej ładunku w łańcuch. Aby utworzyć ładunek tablicy bajtów z poziomu `java.lang.String` lub `java.lang.StringBuffer`, należy użyć metody `getBytes` .

Metoda `getBytes` przekształca łańcuch w domyślny zestaw znaków dla platformy. Domyślnym zestawem znaków jest zazwyczaj `UTF-8`. Publikacje dotyczące produktu MQTT , które zawierają tylko tekst, są zwykle kodowane w produkcie `UTF-8`. Użyj metody `getBytes("UTF8")` , aby przesłonić domyślny zestaw znaków.

W produkcie IBM MQpublikacja MQTT jest odbierana jako komunikat `jms-bytes` . Komunikat zawiera folder `MQRFH2` zawierający `<mqtt>`oraz folder `<mqps>` . Folder `<mqtt>` zawiera elementy `clientId`, `msgId`i `qos`, ale ta treść może ulec zmianie w przyszłości.

`MqttMessage` ma trzy dodatkowe atrybuty: jakość usługi, bez względu na to, czy jest ona zachowywana, oraz czy jest duplikatem. Flaga duplikowania jest ustawiana tylko wtedy, gdy jakość usługi jest "co najmniej raz" lub "dokładnie jeden raz". Jeśli komunikat został wcześniej wysłany i nie został uznany za wystarczająco szybko przez klienta MQTT , komunikat zostanie wysłany ponownie z zduplikowanym atrybutem ustawionym na wartość `true`.

Publikowanie

Aby utworzyć publikację w aplikacji klienckiej produktu MQTT , należy utworzyć `MqttMessage`. Ustaw jego ładunek, jakość usługi i to, czy jest on zachowywany, a następnie wywołaj metodę `MqttTopic.publish(MqttMessage message)` ; `MqttDeliveryToken` jest zwracany, a zakończenie publikacji jest asynchroniczne.

Alternatywnie klient MQTT może utworzyć tymczasowy obiekt komunikatu dla użytkownika z parametrów metody `MqttTopic.publish(byte [] payload, int qos, boolean retained)` podczas tworzenia publikacji.

Jeśli w publikacji znajduje się "co najmniej raz" lub "dokładnie jeden raz" jakość usług, `QoS=1` lub `QoS=2`, klient MQTT wywołuje interfejs `MqttClientPersistence` . Wywołuje on produkt `MqttClientPersistence` w celu zapisania komunikatu przed zwróceniem znacznika dostarczenia do aplikacji.

Aplikacja może zablokować do momentu dostarczenia wiadomości do serwera za pomocą metody `MqttDeliveryToken.waitForCompletion` . Alternatywnie, aplikacja może być kontynuowana bez blokowania. Jeśli chcesz sprawdzić, czy publikacje są dostarczane bez blokowania, zarejestruj instancję klasy zwrotnej, która implementuje `MqttCallback` z klientem MQTT . Klient MQTT wywołuje metodę `MqttCallback.deliveryComplete` od razu po dostarczyniu publikacji. W zależności od jakości usługi dostawa może być niemal natychmiastowa w przypadku produktu `QoS=0`lub może potrwać pewien czas w przypadku produktu `QoS=2`.

Użyj metody `MqttDeliveryToken.isComplete` , aby odpytać, czy dostawa jest kompletna. Jeśli wartością parametru `MqttDeliveryToken.isComplete` jest `false`, można wywołać `MqttDeliveryToken.getMessage` , aby pobrać treść wiadomości. Jeśli wynikiem wywołania programu `MqttDeliveryToken.isComplete` jest `true`, komunikat został usunięty, a wywołanie metody `MqttDeliveryToken.getMessage` spowodowałoby zgłoszenie wyjątku pustego wskaźnika. Nie ma wbudowanej synchronizacji między `MqttDeliveryToken.getMessage` i `MqttDeliveryToken.isComplete`.

Jeśli klient rozłącza się przed odebraniem wszystkich oczekujących tokenów dostarczania, nowa instancja klienta może wysłać zapytanie do tokenów dostarczania oczekujących przed nawiązaniem połączenia. Dopóki klient nie połączy się, nie zostaną wykonane żadne nowe dostawy i można bezpiecznie zadzwonić do produktu `MqttDeliveryToken.getMessage`. Użyj metody `MqttDeliveryToken.getMessage` , aby dowiedzieć się, które publikacje nie zostały dostarczone. Oczekujące tokeny dostarczania są usuwane w przypadku nawiązania połączenia z `MqttConnectOptions.cleanSession` ustawionym na wartość domyślną `true`.

subskrypcja

Menedżer kolejek lub produkt IBM MessageSight jest odpowiedzialny za tworzenie publikacji, które mają zostać wysłane do subskrybenta produktu MQTT . Menedżer kolejek sprawdza, czy filtr tematu

w subskrypcji utworzonej przez klienta MQTT jest zgodny z łańcuchem tematu w publikacji. Zgodność może być dokładnym dopasowaniu lub może zawierać znaki wieloznaczne. Zanim opublikowanie zostanie przekazane do subskrybenta przez menedżera kolejek, menedżer kolejek sprawdza atrybuty tematu powiązane z publikacją. Jest ona zgodna z procedurą wyszukiwania opisaną w sekcji Subskrybowanie łańcucha tematu zawierającego znaki wieloznaczne w celu określenia, czy obiekt tematu administracyjnego nadaje uprawnienia użytkownika do subskrybowania.

Gdy klient MQTT otrzymuje publikację z "co najmniej raz" jakością usługi, wywołuje metodę `MqttCallback.messageArrived` w celu przetworzenia publikacji. Jeśli jakość usługi publikacji to "dokładnie jeden raz", `QoS=2`, klient MQTT wywołuje interfejs `MqttClientPersistence` w celu zapisania komunikatu po jego odebraniu. Następnie wywołuje on `MqttCallback.messageArrived`.

Pojęcia pokrewne

Procedury zwrotne i synchronizacja w aplikacjach klienckich produktu MQTT
W modelu programistycznym klienta produktu MQTT są używane obszernie wątki. Wątki dekodują aplikację kliencką MQTT , o ile tylko mogą, z opóźnień w przesyłaniu komunikatów do serwera i z serwera. Publikacje, tokeny dostarczania i zdarzenia utraconych połączeń są dostarczane do metod w klasie wywołania zwrotnego, która implementuje interfejs `MqttCallback`.

Czyste sesje
Klient MQTT i usługa telemetryczna (MQXR) obsługują informacje o stanie sesji. Informacje o stanie są używane do zapewnienia "co najmniej raz" i "dokładnie raz" dostarczania, a "dokładnie raz" otrzymania publikacji. Stan sesji obejmuje również subskrypcje utworzone przez klienta MQTT . Istnieje możliwość uruchomienia klienta MQTT z zachowaniem informacji o stanie między sesjami lub bez niego. Zmień tryb czyszczenia sesji, ustawiając wartość `MqttConnectOptions.cleanSession` przed nawiązaniem połączenia.

Identyfikator klienta
Identyfikator klienta to 23 bajtowy łańcuch, który identyfikuje klienta MQTT . Każdy identyfikator musi być unikalny w obrębie wszystkich połączonych w danym momencie klientów. Identyfikator musi zawierać tylko znaki poprawne w nazwie menedżera kolejek. W ramach tych ograniczeń można użyć dowolnego łańcucha identyfikacyjnego. Ważne jest, aby procedura przydzielania identyfikatorów klientów była procedurą, a także sposób konfigurowania klienta z wybranym identyfikatorem.

Znaczniki dostawy

Publikacja ostatniego testamentu i testamentu
Jeśli połączenie klienta MQTT nieoczekiwanie kończy się, można skonfigurować MQ Telemetry w taki sposób, aby wysyłał publikację "ostatni testament i testament". Należy wstępnie zdefiniować treść publikacji oraz temat, do którego ma zostać wysłana. Właściwość "last will and testament" jest właściwością połączenia. Utwórz go 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 jeden raz". Można zaimplementować własny mechanizm utrwalania na kliencie lub użyć domyślnego mechanizmu utrwalania udostępnianego wraz z klientem. Trwałość działa w obu kierunkach, w przypadku publikacji wysyłanych do klienta lub z niego.

Jakość usług świadczonych przez klienta MQTT
Klient MQTT udostępnia trzy cechy usługi umożliwiające dostarczanie publikacji do produktu IBM MQ oraz do klienta MQTT : "co najwyżej raz", "co najmniej raz" i "dokładnie jeden raz". Gdy klient MQTT wysyła żądanie do programu IBM MQ w celu utworzenia subskrypcji, żądanie jest wysyłane wraz z jakością usługi co najmniej raz.

Zachowane publikacje i klienty MQTT
Temat może mieć jedną i tylko jedną, zachowaną publikację. Jeśli subskrypcja została utworzona w temacie, który ma zachowaną publikację, publikacja jest natychmiast przekazywana do użytkownika.

Subskrypcje
Utwórz subskrypcje, aby zarejestrować zainteresowanie tematami publikacji przy użyciu filtru tematów. Klient może utworzyć wiele subskrypcji lub subskrypcję zawierającą filtr tematów, który korzysta ze znaków wieloznacznych, w celu zarejestrowania zainteresowania w wielu tematach. Publikacje dotyczące

tematów zgodnych z filtrami są wysyłane do klienta. Subskrypcje mogą pozostać aktywne, gdy klient jest odłączony. Publikacje są wysyłane do klienta po ponownym nawiązaniu połączenia.

Łańcuchy tematów i filtry tematów w klientach MQTT
Łańcuch tematu i filtry tematów są używane do publikowania i subskrybowania. Składnia łańcuchów tematów i filtrów w klientach MQTT jest w dużej mierze taka sama, jak łańcuchy tematów w produkcie IBM MQ.

Jakość usług świadczonych przez klienta MQTT

Klient MQTT udostępnia trzy cechy usługi umożliwiające dostarczanie publikacji do produktu IBM MQ oraz do klienta MQTT : "co najwyżej raz", "co najmniej raz" i "dokładnie jeden raz". Gdy klient MQTT wysyła żądanie do programu IBM MQ w celu utworzenia subskrypcji, żądanie jest wysyłane wraz z jakością usługi co najmniej raz.

Jakość usługi publikacji jest atrybutem produktu `MqttMessage`. Jest ona ustawiana przy użyciu metody `MqttMessage.setQos`.

Metoda `MqttClient.subscribe` może zmniejszyć jakość usługi stosowaną do publikacji wysyłanych do klienta w temacie. Jakość usługi publikacji przekazanej abonentowi może być inna niż jakość usługi w publikacji. Niższa z tych dwóch wartości jest używana do przekazywania publikacji.

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 zapisany. Komunikat może zostać utracony, jeśli klient zostanie rozłączony lub nastąpi awaria serwera.

`QoS=0` jest najszybszym trybem przesyłania. Czasami nazywa się "ogień i zapomnij".

Serwer MQTT protocol nie wymaga od serwerów przekazywania publikacji pod adresem `QoS=0` do klienta. Jeśli klient zostanie odłączony od chwili otrzymania przez serwer publikacji, może ona zostać odrzucona w zależności od serwera. Usługa telemetryczna (MQXR) nie usuwa komunikatów wysłanych z produktem `QoS=0`. Są one zapisywane jako komunikaty nietrwałe i są usuwane tylko w przypadku zatrzymania menedżera kolejek.

Co najmniej raz
`QoS=1`

`QoS=1` jest domyślnym trybem przesyłania.

Komunikat jest zawsze dostarczany co najmniej raz. Jeśli nadawca nie otrzyma potwierdzenia, komunikat zostanie wysłany ponownie z ustawioną opcją `DUP` , dopóki nie zostanie odebrany potwierdzenie. W wyniku tego, odbiorca może wielokrotnie wysyłać ten sam komunikat i może przetwarzać go wielokrotnie.

Komunikat musi być przechowywany lokalnie u nadawcy i odbiorcy, dopóki nie zostanie przetworzony.

Komunikat zostanie usunięty z odbiornika po przetworzeniu komunikatu. Jeśli odbiornik jest brokerem, komunikat jest publikowany do jego subskrybentów. Jeśli odbiorcą jest klient, komunikat jest dostarczany do aplikacji subskrybenta. Po usunięciu wiadomości odbiorca wysyła potwierdzenie do nadawcy.

Wiadomość zostanie usunięta od nadawcy po odebraniu potwierdzenia od odbiorcy.

Dokładnie jeden raz
`QoS=2`

Komunikat jest zawsze dostarczany dokładnie jeden raz.

Komunikat musi być przechowywany lokalnie u nadawcy i odbiorcy, dopóki nie zostanie przetworzony.

`QoS=2` jest najbezpieczniejszym, ale najwolniejszym trybem przesyłania. Przed usunięciem komunikatu z nadajnika pobiera ona co najmniej dwie pary transmisji między nadawcą i odbiorcą. Komunikat może być przetwarzany u odbiorcy po pierwszej transmisji.

W pierwszej parze transmisji nadawca przesyła wiadomość i otrzymuje potwierdzenie od odbiorcy, że zapisał komunikat. Jeśli nadawca nie otrzyma potwierdzenia, komunikat zostanie wysłany ponownie z ustawioną opcją DUP , dopóki nie zostanie odebrany potwierdzenie.

W drugiej parze transmisji nadawca informuje odbiorcę o tym, że może zakończyć przetwarzanie komunikatu "PUBREL". Jeśli nadawca nie otrzyma potwierdzenia komunikatu "PUBREL" , komunikat "PUBREL" zostanie wysłany ponownie, dopóki nie zostanie odebrany potwierdzenie. Nadawca usuwa komunikat, który został zapisany po odebraniu potwierdzenia do komunikatu produktu "PUBREL" .

Odbiornik może przetworzyć komunikat w pierwszej lub drugiej fazie, pod warunkiem, że nie przeprzetwarzali komunikatu. Jeśli odbiornik jest brokerem, publikuje on komunikat do subskrybentów. Jeśli odbiorcą jest klient, dostarcza on komunikat do aplikacji subskrybenta. Odbiornik wysyła komunikat o zakończeniu z powrotem do nadawcy, który zakończył przetwarzanie komunikatu.

Pojęcia pokrewne

Procedury zwrotne i synchronizacja w aplikacjach klienckich produktu MQTT
W modelu programistycznym klienta produktu MQTT są używane obszernie wątki. Wątki dekodują aplikację kliencką MQTT , o ile tylko mogą, z opóźnień w przesyłaniu komunikatów do serwera i z serwera. Publikacje, tokeny dostarczania i zdarzenia utraconych połączeń są dostarczane do metod w klasie wywołania zwrotnego, która implementuje interfejs `MqttCallback`.

Czyste sesje
Klient MQTT i usługa telemetryczna (MQXR) obsługują informacje o stanie sesji. Informacje o stanie są używane do zapewnienia "co najmniej raz" i "dokładnie raz" dostarczania, a "dokładnie raz" otrzymania publikacji. Stan sesji obejmuje również subskrypcje utworzone przez klienta MQTT . Istnieje możliwość uruchomienia klienta MQTT z zachowaniem informacji o stanie między sesjami lub bez niego. Zmień tryb czyszczenia sesji, ustawiając wartość `MqttConnectOptions.cleanSession` przed nawiązaniem połączenia.

Identyfikator klienta
Identyfikator klienta to 23 bajtowy łańcuch, który identyfikuje klienta MQTT . Każdy identyfikator musi być unikalny w obrębie wszystkich połączonych w danym momencie klientów. Identyfikator musi zawierać tylko znaki poprawne w nazwie menedżera kolejek. W ramach tych ograniczeń można użyć dowolnego łańcucha identyfikacyjnego. Ważne jest, aby procedura przydzielania identyfikatorów klientów była procedurą, a także sposób konfigurowania klienta z wybranym identyfikatorem.

Znaczniki dostawy

Publikacja ostatniego testamentu i testamentu
Jeśli połączenie klienta MQTT nieoczekiwanie kończy się, można skonfigurować MQ Telemetry w taki sposób, aby wysyłał publikację "ostatni testament i testament". Należy wstępnie zdefiniować treść publikacji oraz temat, do którego ma zostać wysłana. Właściwość "last will and testament" jest właściwością połączenia. Utwórz go 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 jeden raz". Można zaimplementować własny mechanizm utrwalania na kliencie lub użyć domyślnego mechanizmu utrwalania udostępnianego wraz z klientem. Trwałość działa w obu kierunkach, w przypadku publikacji wysyłanych do klienta lub z niego.

Publikacje
Publikacje to instancje produktu `MqttMessage` , które są powiązane z łańcuchem tematu. Klienty produktu MQTT mogą tworzyć publikacje, które mają być wysyłane do produktu IBM MQ, a następnie zasubskrybować tematy w produkcie IBM MQ , aby otrzymywać publikacje.

Zachowane publikacje i klienty MQTT
Temat może mieć jedną i tylko jedną, zachowaną publikację. Jeśli subskrypcja została utworzona w temacie, który ma zachowaną publikację, publikacja jest natychmiast przekazywana do użytkownika.

Subskrypcje
Utwórz subskrypcje, aby zarejestrować zainteresowanie tematami publikacji przy użyciu filtru tematów. Klient może utworzyć wiele subskrypcji lub subskrypcję zawierającą filtr tematów, który korzysta ze znaków wieloznacznych, w celu zarejestrowania zainteresowania w wielu tematach. Publikacje dotyczące

tematów zgodnych z filtrami są wysyłane do klienta. Subskrypcje mogą pozostać aktywne, gdy klient jest odłączony. Publikacje są wysyłane do klienta po ponownym nawiązaniu połączenia.

Łańcuchy tematów i filtry tematów w klientach MQTT
Łańcuch tematu i filtry tematów są używane do publikowania i subskrybowania. Składnia łańcuchów tematów i filtrów w klientach MQTT jest w dużej mierze taka sama, jak łańcuchy tematów w produkcie IBM MQ.

Zachowane publikacje i klienty MQTT

Temat może mieć jedną i tylko jedną, zachowaną publikację. Jeśli subskrypcja została utworzona w temacie, który ma zachowaną publikację, publikacja jest natychmiast przekazywana do użytkownika.

Metoda `MqttMessage.setRetained` służy do określania, czy publikacja na danym temacie jest zachowywana.

Po utworzeniu lub zaktualizowaniu zachowanej publikacji należy wysłać publikację za pomocą opcji QoS 1 lub 2. Jeśli zostanie ona wysłana z QoS z 0, program IBM MQ tworzy nietrwałą zachowaną publikację. Jeśli menedżer kolejek zostanie zatrzymany, publikacja nie zostanie zachowana.

Jeśli opublikujesz niezachowaną publikację do tematu, który ma zachowaną publikację, zachowana publikacja nie będzie miała wpływu na tę publikację. Aktualni subskrybenci otrzymują nową publikację. Nowi subskrybenci otrzymują po raz pierwszy zachowaną publikację, a następnie otrzymują nowe publikacje.

Zachowaną publikację można użyć do zarejestrowania najnowszej wartości pomiaru. Nowi subskrybenci tematu otrzymują od razu najnowszą wartość pomiaru. Jeśli nie są pobierane żadne nowe pomiary od czasu ostatniego subskrybowania subskrybenta do tematu publikacji, a subskrybent ponownie subskrybuje subskrybent, subskrybent otrzymuje najnowszą zachowaną publikację w temacie ponownie.

Aby usunąć zachowaną publikację, dostępne są dwie opcje:

- Uruchom komendę MQSC **`CLEAR TOPICSTR`** .
- Utwórz zachowaną publikację o zerowej długości. Jak określono w specyfikacji MQTT 3.1.1 , jeśli zachowany komunikat o zerowej długości jest publikowany w temacie, wszystkie zachowane komunikaty dla tego tematu zostaną wyczyszczone.

Pojęcia pokrewne

Procedury zwrotne i synchronizacja w aplikacjach klienckich produktu MQTT
W modelu programistycznym klienta produktu MQTT są używane obszernie wątki. Wątki dekodują aplikację kliencką MQTT , o ile tylko mogą, z opóźnień w przesyłaniu komunikatów do serwera i z serwera. Publikacje, tokeny dostarczania i zdarzenia utraconych połączeń są dostarczane do metod w klasie wywołania zwrotnego, która implementuje interfejs `MqttCallback`.

Czyste sesje
Klient MQTT i usługa telemetryczna (MQXR) obsługują informacje o stanie sesji. Informacje o stanie są używane do zapewnienia "co najmniej raz" i "dokładnie raz" dostarczania, a "dokładnie raz" otrzymania publikacji. Stan sesji obejmuje również subskrypcje utworzone przez klienta MQTT . Istnieje możliwość uruchomienia klienta MQTT z zachowaniem informacji o stanie między sesjami lub bez niego. Zmień tryb czyszczenia sesji, ustawiając wartość `MqttConnectOptions.cleanSession` przed nawiązaniem połączenia.

Identyfikator klienta
Identyfikator klienta to 23 bajtowy łańcuch, który identyfikuje klienta MQTT . Każdy identyfikator musi być unikalny w obrębie wszystkich połączonych w danym momencie klientów. Identyfikator musi zawierać tylko znaki poprawne w nazwie menedżera kolejek. W ramach tych ograniczeń można użyć dowolnego łańcucha identyfikacyjnego. Ważne jest, aby procedura przydzielania identyfikatorów klientów była procedurą, a także sposób konfigurowania klienta z wybranym identyfikatorem.

Znaczniki dostawy

Publikacja ostatniego testamentu i testamentu
Jeśli połączenie klienta MQTT nieoczekiwanie kończy się, można skonfigurować MQ Telemetry w taki sposób, aby wysyłał publikację "ostatni testament i testament". Należy wstępnie zdefiniować treść

publikacji oraz temat, do którego ma zostać wysłana. Właściwość "last will and testament" jest właściwością połączenia. Utwórz go 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 jeden raz". Można zaimplementować własny mechanizm utrwalania na kliencie lub użyć domyślnego mechanizmu utrwalania udostępnianego wraz z klientem. Trwałość działa w obu kierunkach, w przypadku publikacji wysyłanych do klienta lub z niego.

Publikacje
Publikacje to instancje produktu `MqttMessage` , które są powiązane z łańcuchem tematu. Klienty produktu MQTT mogą tworzyć publikacje, które mają być wysyłane do produktu IBM MQ, a następnie zasubskrybować tematy w produkcie IBM MQ , aby otrzymywać publikacje.

Jakość usług świadczonych przez klienta MQTT
Klient MQTT udostępnia trzy cechy usługi umożliwiające dostarczanie publikacji do produktu IBM MQ oraz do klienta MQTT : "co najwyżej raz", "co najmniej raz" i "dokładnie jeden raz". Gdy klient MQTT wysyła żądanie do programu IBM MQ w celu utworzenia subskrypcji, żądanie jest wysyłane wraz z jakością usługi co najmniej raz.

Subskrypcje
Utwórz subskrypcje, aby zarejestrować zainteresowanie tematami publikacji przy użyciu filtru tematów. Klient może utworzyć wiele subskrypcji lub subskrypcję zawierającą filtr tematów, który korzysta ze znaków wieloznacznych, w celu zarejestrowania zainteresowania w wielu tematach. Publikacje dotyczące tematów zgodnych z filtrami są wysyłane do klienta. Subskrypcje mogą pozostać aktywne, gdy klient jest odłączony. Publikacje są wysyłane do klienta po ponownym nawiązaniu połączenia.

Łańcuchy tematów i filtry tematów w klientach MQTT
Łańcuch tematu i filtry tematów są używane do publikowania i subskrybowania. Składnia łańcuchów tematów i filtrów w klientach MQTT jest w dużej mierze taka sama, jak łańcuchy tematów w produkcie IBM MQ.

Subskrypcje

Utwórz subskrypcje, aby zarejestrować zainteresowanie tematami publikacji przy użyciu filtru tematów. Klient może utworzyć wiele subskrypcji lub subskrypcję zawierającą filtr tematów, który korzysta ze znaków wieloznacznych, w celu zarejestrowania zainteresowania w wielu tematach. Publikacje dotyczące tematów zgodnych z filtrami są wysyłane do klienta. Subskrypcje mogą pozostać aktywne, gdy klient jest odłączony. Publikacje są wysyłane do klienta po ponownym nawiązaniu połączenia.

Utwórz subskrypcje przy użyciu metod `MqttClient.subscribe` , przekazując jeden lub więcej filtrów tematów oraz parametry jakości usług. Parametr jakości usługi ustawia maksymalną jakość usługi, którą subskrybent jest przygotowany do użycia w celu odebrania komunikatu. Wiadomości wysłane do tego klienta nie mogą być dostarczane z wyższą jakością usług. Jakość usługi jest ustawiana na niższą od pierwotnej wartości, gdy komunikat został opublikowany, a poziom określony dla subskrypcji. Domyślną jakością usługi dla odbierania komunikatów jest `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 także łańcuch tematu, z którym komunikat został opublikowany w subskrybencie.

Za pomocą metod `MqttClient.unsubscribe` można usunąć subskrypcję lub 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, które są zgodne z aktywnymi subskrypcjami, są wysyłane do klienta natychmiast po ich opublikowaniu. Jeśli klient zostanie rozłączony, zostaną one wysłane do klienta, jeśli połączy się ponownie z tym samym serwerem z tym samym identyfikatorem klienta i z `MqttConnectOptions.cleanSession` ustawionym na `false`.

Subskrypcje dla konkretnego klienta są identyfikowane przez identyfikator klienta. Można ponownie podłączyć klient z innego urządzenia klienckiego do tego samego serwera i kontynuować te same subskrypcje i otrzymywać niedostarczone publikacje.

Pojęcia pokrewne

Procedury zwrotne i synchronizacja w aplikacjach klienckich produktu MQTT
W modelu programistycznym klienta produktu MQTT są używane obszernie wątki. Wątki dekodują aplikację kliencką MQTT , o ile tylko mogą, z opóźnień w przesyłaniu komunikatów do serwera i z serwera. Publikacje, tokeny dostarczania i zdarzenia utraconych połączeń są dostarczane do metod w klasie wywołania zwrotnego, która implementuje interfejs `MqttCallback`.

Czyste sesje
Klient MQTT i usługa telemetryczna (MQXR) obsługują informacje o stanie sesji. Informacje o stanie są używane do zapewnienia "co najmniej raz" i "dokładnie raz" dostarczania, a "dokładnie raz" otrzymania publikacji. Stan sesji obejmuje również subskrypcje utworzone przez klienta MQTT . Istnieje możliwość uruchomienia klienta MQTT z zachowaniem informacji o stanie między sesjami lub bez niego. Zmień tryb czyszczenia sesji, ustawiając wartość `MqttConnectOptions.cleanSession` przed nawiązaniem połączenia.

Identyfikator klienta
Identyfikator klienta to 23 bajtowy łańcuch, który identyfikuje klienta MQTT . Każdy identyfikator musi być unikalny w obrębie wszystkich połączonych w danym momencie klientów. Identyfikator musi zawierać tylko znaki poprawne w nazwie menedżera kolejek. W ramach tych ograniczeń można użyć dowolnego łańcucha identyfikacyjnego. Ważne jest, aby procedura przydzielania identyfikatorów klientów była procedurą, a także sposób konfigurowania klienta z wybranym identyfikatorem.

Znaczniki dostawy

Publikacja ostatniego testamentu i testamentu
Jeśli połączenie klienta MQTT nieoczekiwanie kończy się, można skonfigurować MQ Telemetry w taki sposób, aby wysyłał publikację "ostatni testament i testament". Należy wstępnie zdefiniować treść publikacji oraz temat, do którego ma zostać wysłana. Właściwość "last will and testament" jest właściwością połączenia. Utwórz go 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 jeden raz". Można zaimplementować własny mechanizm utrwalania na kliencie lub użyć domyślnego mechanizmu utrwalania udostępnianego wraz z klientem. Trwałość działa w obu kierunkach, w przypadku publikacji wysyłanych do klienta lub z niego.

Publikacje

Publikacje to instancje produktu `MqttMessage` , które są powiązane z łańcuchem tematu. Klienty produktu MQTT mogą tworzyć publikacje, które mają być wysyłane do produktu IBM MQ, a następnie zasubskrybować tematy w produkcie IBM MQ , aby otrzymywać publikacje.

Jakość usług świadczonych przez klienta MQTT
Klient MQTT udostępnia trzy cechy usługi umożliwiające dostarczanie publikacji do produktu IBM MQ oraz do klienta MQTT : "co najwyżej raz", "co najmniej raz" i "dokładnie jeden raz". Gdy klient MQTT wysyła żądanie do programu IBM MQ w celu utworzenia subskrypcji, żądanie jest wysyłane wraz z jakością usługi co najmniej raz.

Zachowane publikacje i klienty MQTT
Temat może mieć jedną i tylko jedną, zachowaną publikację. Jeśli subskrypcja została utworzona w temacie, który ma zachowaną publikację, publikacja jest natychmiast przekazywana do użytkownika.

Łańcuchy tematów i filtry tematów w klientach MQTT
Łańcuch tematu i filtry tematów są używane do publikowania i subskrybowania. Składnia łańcuchów tematów i filtrów w klientach MQTT jest w dużej mierze taka sama, jak łańcuchy tematów w produkcie IBM MQ.

Łańcuchy tematów i filtry tematów w klientach MQTT

Łańcuch tematu i filtry tematów są używane do publikowania i subskrybowania. Składnia łańcuchów tematów i filtrów w klientach MQTT jest w dużej mierze taka sama, jak łańcuchy tematów w produkcie 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 są używane do subskrybowania tematów i otrzymywania publikacji. Filtry tematów mogą zawierać znaki wieloznaczne. W przypadku znaków wieloznacznych można zasubskrybować wiele tematów. Aby utworzyć filtr tematów, należy użyć metody subskrypcji, na przykład `MqttClient.subscribe(java.lang.String topicFilter)`.

Łańcuchy tematów

Składnia łańcucha tematu IBM MQ jest opisana w sekcji Strings tematów. Składnia łańcuchów tematów 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 programu IBM MQ przez klienty MQTT muszą być zgodne z konwencją dla nazw menedżerów kolejek.
2. Maksymalna długość różni się od siebie. Łańcuchy tematów IBM MQ są ograniczone do 10,240 znaków. Klient MQTT może tworzyć łańcuchy tematów o długości do 65535 bajtów.
3. Łańcuch tematu utworzony przez klient 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 publikowania/subskrybowania produktu IBM MQ protokół `mqttv3` nie zawiera pojęcia administracyjnego obiektu tematu. Nie można skonstruować łańcucha tematu z obiektu tematu i łańcucha tematu. Jednak łańcuch tematu jest odwzorowywany na temat administracyjny w produkcie IBM MQ. Kontrola dostępu powiązana z tematem administracyjnym określa, czy publikacja jest publikowana w temacie, czy odrzucona. Atrybuty, które są stosowane do publikacji, gdy są przekazywane do subskrybentów, mają wpływ na atrybuty tematu administracyjnego.

Filtry tematów

Składnia filtru tematu produktu IBM MQ jest opisana w sekcji Schemat znaków wieloznacznych oparty na temacie. Składnia filtrów tematów, które można skonstruować 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

Procedury zwrotne i synchronizacja w aplikacjach klienckich produktu MQTT
W modelu programistycznym klienta produktu MQTT są używane obszernie wątki. Wątki dekodują aplikację kliencką MQTT , o ile tylko mogą, z opóźnień w przesyłaniu komunikatów do serwera i z serwera. Publikacje, tokeny dostarczania i zdarzenia utraconych połączeń są dostarczane do metod w klasie wywołania zwrotnego, która implementuje interfejs `MqttCallback`.

Czyste sesje
Klient MQTT i usługa telemetryczna (MQXR) obsługują informacje o stanie sesji. Informacje o stanie są używane do zapewnienia "co najmniej raz" i "dokładnie raz" dostarczania, a "dokładnie raz" otrzymania publikacji. Stan sesji obejmuje również subskrypcje utworzone przez klienta MQTT . Istnieje możliwość uruchomienia klienta MQTT z zachowaniem informacji o stanie między sesjami lub bez niego. Zmień tryb czyszczenia sesji, ustawiając wartość `MqttConnectOptions.cleanSession` przed nawiązaniem połączenia.

Identyfikator klienta
Identyfikator klienta to 23 bajtowy łańcuch, który identyfikuje klienta MQTT . Każdy identyfikator musi być unikalny w obrębie wszystkich połączonych w danym momencie klientów. Identyfikator musi zawierać tylko znaki poprawne w nazwie menedżera kolejek. W ramach tych ograniczeń można użyć dowolnego łańcucha identyfikacyjnego. Ważne jest, aby procedura przydzielania identyfikatorów klientów była procedurą, a także sposób konfigurowania klienta z wybranym identyfikatorem.

Znaczniki dostawy

Publikacja ostatniego testamentu i testamentu
Jeśli połączenie klienta MQTT nieoczekiwanie kończy się, można skonfigurować MQ Telemetry w taki sposób, aby wysyłał publikację "ostatni testament i testament". Należy wstępnie zdefiniować treść publikacji oraz temat, do którego ma zostać wysłana. Właściwość "last will and testament" jest właściwością połączenia. Utwórz go 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 jeden raz". Można zaimplementować własny mechanizm utrwalania na kliencie lub użyć domyślnego mechanizmu utrwalania udostępnianego wraz z klientem. Trwałość działa w obu kierunkach, w przypadku publikacji wysyłanych do klienta lub z niego.

Publikacje
Publikacje to instancje produktu `MqttMessage` , które są powiązane z łańcuchem tematu. Klienty produktu MQTT mogą tworzyć publikacje, które mają być wysyłane do produktu IBM MQ, a następnie zasubskrybować tematy w produkcie IBM MQ , aby otrzymywać publikacje.

Jakość usług świadczonych przez klienta MQTT
Klient MQTT udostępnia trzy cechy usługi umożliwiające dostarczanie publikacji do produktu IBM MQ oraz do klienta MQTT : "co najwyżej raz", "co najmniej raz" i "dokładnie jeden raz". Gdy klient MQTT wysyła żądanie do programu IBM MQ w celu utworzenia subskrypcji, żądanie jest wysyłane wraz z jakością usługi co najmniej raz.

Zachowane publikacje i klienty MQTT
Temat może mieć jedną i tylko jedną, zachowaną publikację. Jeśli subskrypcja została utworzona w temacie, który ma zachowaną publikację, publikacja jest natychmiast przekazywana do użytkownika.

Subskrypcje
Utwórz subskrypcje, aby zarejestrować zainteresowanie tematami publikacji przy użyciu filtru tematów. Klient może utworzyć wiele subskrypcji lub subskrypcję zawierającą filtr tematów, który korzysta ze znaków wieloznacznych, w celu zarejestrowania zainteresowania w wielu tematach. Publikacje dotyczące tematów zgodnych z filtrami są wysyłane do klienta. Subskrypcje mogą pozostać aktywne, gdy klient jest odłączony. Publikacje są wysyłane do klienta po ponownym nawiązaniu połączenia.

Tworzenie aplikacji Microsoft Windows Communication Foundation przy użyciu produktu IBM MQ

Niestandardowy kanał programu Microsoft Windows Communication Foundation (WCF) dla IBM MQ wysyła i odbiera komunikaty między klientami i usługami WCF.

Pojęcia pokrewne

"Wprowadzenie do niestandardowego kanału produktu IBM MQ dla systemu WCF z produktem .NET" na stronie 1379
Kanał niestandardowy dla produktu IBM MQ jest kanałem transportowym, korzystając z ujednoliconego modelu programistycznego produktu Microsoft Windows Communication Foundation (WCF).

"Korzystanie z niestandardowych kanałów produktu IBM MQ dla produktu WCF" na stronie 1384
Przegląd informacji dostępnych dla programistów korzystających z niestandardowych kanałów produktu IBM MQ dla programu Windows Communication Foundation (WCF).

"Korzystanie z przykładów WCF" na stronie 1404
Przykłady produktu Windows Communication Foundation (WCF) zawierają kilka prostych przykładów użycia niestandardowego kanału produktu IBM MQ .

FFST: technologia WCF XMS First Failure Support Technology

Zadania pokrewne

Śledzenie niestandardowego kanału WCF dla produktu IBM MQ

Rozwiązywanie problemów z kanałem niestandardowym WCF w przypadku problemów z produktem IBM MQ

Wprowadzenie do niestandardowego kanału produktu IBM MQ dla systemu WCF z produktem .NET

Kanał niestandardowy dla produktu IBM MQ jest kanałem transportowym, korzystając z ujednoliconego modelu programistycznego produktu Microsoft Windows Communication Foundation (WCF).

Środowisko Microsoft Windows Communication Foundation, wprowadzone w programie Microsoft.NET 3, umożliwia tworzenie aplikacji i usług .NET niezależnie od transportu i protokołów używanych do ich łączenia, co umożliwia użycie alternatywnych transportów lub konfiguracji w zależności od środowiska, w którym wdrożono usługę lub aplikację.

Połączenia są zarządzane w czasie wykonywania przez system WCF, budując stos kanałów zawierający wymaganą kombinację następujących elementów:

- Elementy protokołu: opcjonalny zestaw elementów, w których żaden, jeden lub więcej nie może być dodany do protokołów obsługi, takich jak normy WS-*.
- Koder komunikatów: obowiązkowy element w stosie sterujący serializacją komunikatu do jego formatu łącznika.
- Kanał transportowy: obowiązkowy element w stosie odpowiedzialny za transport serializowanego komunikatu do jego punktu końcowego.

Kanał niestandardowy dla produktu IBM MQ jest kanałem transportowym i jako taki musi być sparowany z koderem komunikatów i opcjonalnymi protokołami, które są wymagane przez aplikację przy użyciu niestandardowego powiązania WCF. W ten sposób aplikacje, które zostały opracowane w celu użycia narzędzia WCF, mogą używać kanału niestandardowego dla 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, skalowalnym i niezawodnym przesyłaniem komunikatów produktu IBM MQ. Pełną listę obsługiwanych funkcji można znaleźć pod adresem: "Niestandardowe funkcje i możliwości kanału WCF" na stronie 1384.

Kiedy i dlaczego w systemie WCF używam niestandardowego kanału IBM MQ ?

Niestandardowego kanału produktu IBM MQ można używać do wysyłania i odbierania komunikatów między klientami i usługami WCF w taki sam sposób, jak wbudowane transporty udostępniane przez produkt Microsoft, co umożliwia aplikacjom uzyskiwanie dostępu do funkcji produktu IBM MQ w ramach ujednoliconego modelu programistycznego WCF.

Typowe scenariusze wzorców użycia dla niestandardowego kanału produktu IBM MQ dla WCF to interfejs inny niż SOAP, który umożliwia przesyłanie rodzimych komunikatów produktu IBM MQ .

Komunikaty, które są przenoszone przy użyciu formatu komunikatu innego niż SOAP/Non-JMS (Pure MQMessage)

Jeśli do przesyłania rodzimych komunikatów produktu IBM MQ używany jest niestandardowy kanał produktu IBM MQ dla systemu WCF, komunikaty są przenoszone za pomocą formatu komunikatu innego niż SOAP/Non-JMS (Pure MQMessage) produktu IBM MQ.

Użytkownicy WCF mogą uruchomić usługę lub innymi słowy, użytkownicy usług mogą wysłać komunikat do kolejki produktu IBM MQ za pomocą komunikatów MQMessages. Aplikacje mogą uzyskać i ustawić pola MQMD oraz ładunek. Jeśli komunikat jest dostępny w kolejkach produktu IBM MQ MQ , ten komunikat może być przetwarzany przez dowolną usługę WCF lub aplikacje inne niż WCF, takie jak C lub Java , które działają w systemach Windows, UNIX lub z/OS.

Wymagania dotyczące oprogramowania dla niestandardowego kanału produktu IBM MQ dla systemu WCF

W tym temacie przedstawiono wymagania dotyczące oprogramowania dla niestandardowego kanału produktu IBM MQ dla systemu WCF. Kanał niestandardowy produktu IBM MQ dla systemu WCF może łączyć się tylko z menedżerami kolejek produktu IBM WebSphere MQ 7.0 lub wyższymi.

Wymagania dotyczące środowiska wykonawczego

- Produkt Microsoft.NET Framework v4.5.1 lub nowszy musi być zainstalowany na komputerze hosta.
- Produkty *Java i .NET Messaging and Web Services* są instalowane domyślnie jako część instalatora produktu IBM MQ . Ten komponent służy do instalowania zespołów produktu .NET wymaganych przez kanał niestandardowy do pamięci podręcznej zespołu globalnego.

Uwaga: Jeśli produkt Microsoft .NET Framework V4.5.1 lub nowszy nie jest zainstalowany przed zainstalowaniem produktu IBM MQ, instalacja produktu IBM MQ będzie kontynuowana bez błędów, ale produkt IBM MQ classes for .NET nie jest dostępny. Jeśli produkt.NET Framework jest zainstalowany po zainstalowaniu produktu IBM MQ, zespoły IBM MQ.NET muszą być zarejestrowane, uruchamiając skrypt *WMQInstallDir*`\bin\amqiRegisterdotNet.cmd` , gdzie *WMQInstallDir* to katalog, w którym zainstalowano produkt IBM MQ . Ten skrypt zainstaluje wymagane zespoły w globalnej pamięci podręcznej zespołu (Global Assembly Cache-GAC). W katalogu %TEMP% tworzony jest zestaw plików produktu `amqi*.log` , które rejestrują działania, które zostały wykonane. Nie jest konieczne ponowne uruchomienie skryptu `amqiRegisterdotNet.cmd` , jeśli produkt .NET jest aktualizowany do wersji v4.5.1 lub nowszej z wcześniejszej wersji, na przykład z wersji .NET V3.5.

Wymagania dotyczące środowiska programistycznego

- Microsoft Visual Studio 2008 lub Windows Software Development Kit dla produktu .NET 4.5.1 lub nowszego.
- Produkt Microsoft.NET Framework V4.5.1 lub nowszy musi być zainstalowany na komputerze hosta w celu zbudowania przykładowych plików rozwiązania.

Kanał niestandardowy produktu IBM MQ dla WCF: Co jest zainstalowane?

Kanał niestandardowy dla produktu IBM MQ jest kanałem transportowym, korzystając z ujednoliconego modelu programistycznego produktu Microsoft Windows Communication Foundation (WCF). Kanał niestandardowy jest instalowany domyślnie jako część instalacji.

Kanał niestandardowy produktu IBM MQ dla WCF

Kanał niestandardowy i jego zależności są zawarte w komponencie `Java and .NET Messaging and Web Services` , który jest instalowany domyślnie. W przypadku aktualizowania produktu IBM MQ z wcześniejszej wersji produktuIBM MQ 8.0aktualizacja domyślnie instaluje kanał niestandardowy produktu IBM MQ dla WCF, jeśli komponent `Java and .NET Messaging and Web Services` był wcześniej zainstalowany we wcześniejszej instalacji.

Komponent `.NET Messaging and Web Services` zawiera plik `IBM.XMS.WCF.dll` oraz plik `IBM.WMQ.WCF.dll` , a te pliki są głównym niestandardowym zespołem kanałów, który zawiera klasy interfejsu WCF. Pliki te są instalowane w globalnej pamięci podręcznej zespołu (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 kluczowych, które są wymagane do używania kanału niestandardowego.

Tabela 196. Klasy klucza wymagane do korzystania z kanału niestandardowego

	Interfejs SOAP/JMS (istniejący)	**Interfejs inny niż SOAP/Non-JMS (z produktu IBM MQ 8.0)**
Niestandardowe zespół kanałów	IBM.XMS.WCF.dll	IBM.WMQ.WCF.dll
Nazwa powiązania transportu	IBM.XMS.WCF.SoapJmsIbmTransportBindingElement	IBM.WMQ.WCF.WmqIbmTransportBindingElement
Importer powiązania transportu	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementImporter	IBM.WMQ.WCF.WmqIbmTransportBindingElementImporter
Konfiguracja powiązania transportu	IBM.XMS.WCF.SoapJmsIbmTransportBindingElementConfig	IBM.WMQ.WCF.WmqIbmTransportBindingElementConfig
Przykłady (Oneway)	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 interfejsy SOAP/JMS i Non-SOAP/Non-JMS . W przypadku nowych aplikacji zalecane jest korzystanie z IBM.WMQ.WCF , ponieważ obsługuje oba interfejsy.

Wysyłanie sformatowanych komunikatów MQSTR

V 9.1.0

Jeśli komunikat żądania jest typu MQSTR, można wybrać opcję wysłania komunikatu 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 przypadku wartości **replyMessageFormat** wielkość liter nie jest rozróżniana.
2. Użycie dowolnej wartości innej niż "" lub *MQSTR*powoduje wystąpienie wyjątku o niepoprawnej wartości parametru.

Przykłady niestandardowych kanałów produktu IBM MQ

Przykłady zawierają kilka prostych przykładów użycia niestandardowego kanału produktu 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 przykładów kanału niestandardowego produktu IBM MQ zawiera sekcja “Korzystanie z przykładów WCF” na stronie 1404.

svcutil.exe.config

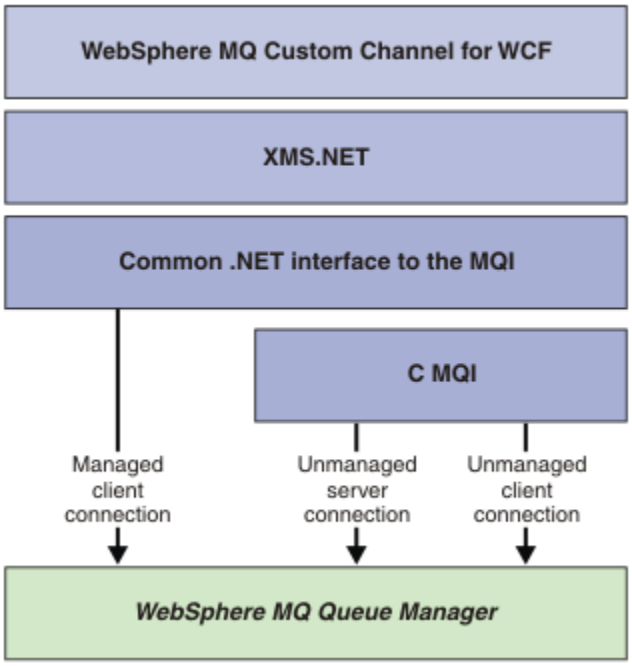
The `svcutil.exe.config` is an example of the configuration settings required to enable the Microsoft WCF `svcutil` client proxy generation tool to recognize the custom channel. Plik `svcutil.exe.config` znajduje się w katalogu *MQ_INSTALLATION_PATH* `\tools\wcf\docs\examples\` , gdzie *MQ_INSTALLATION_PATH* jest katalogiem instalacyjnym produktu IBM MQ. Więcej informacji na temat korzystania z `svcutil.exe.config`zawiera sekcja “Generowanie proxy klienta WCF i plików konfiguracyjnych aplikacji przy użyciu narzędzia svcutil z metadanymi pochodzącym z uruchomionej usługi” na stronie 1401.

Architektura WCF

Niestandardowy kanał produktu IBM MQ dla systemu WCF jest zintegrowany z interfejsem API produktu `IBM Message Service Client for .NET (XMS .NET)`.

Interfejs SOAP/JMS

Architektura WCF jest przedstawiona na poniższym diagramie:



Rysunek 154. Architektura WCF dla interfejsu SOAP/JMS

Wszystkie wymagane komponenty są instalowane domyślnie z instalacją produktu.

Dostępne są następujące trzy połączenia:

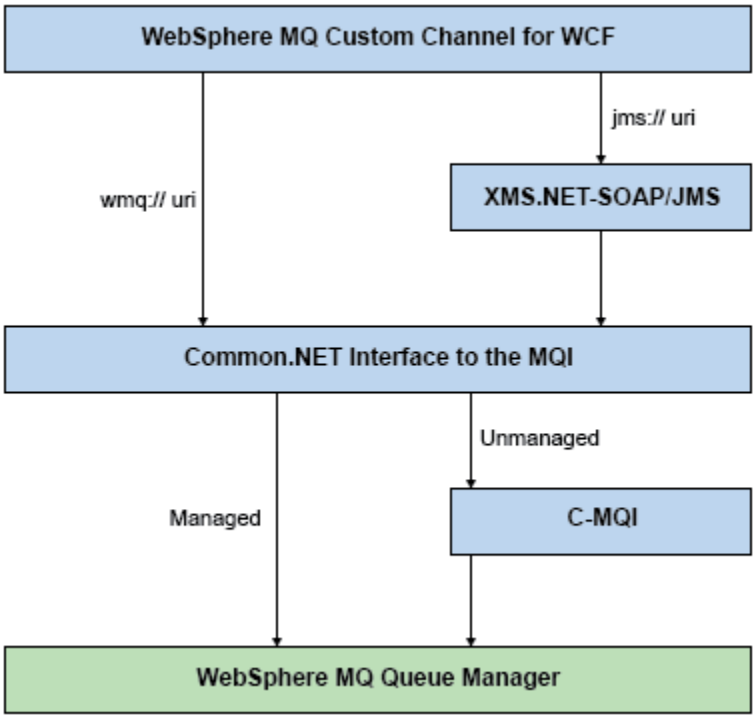
- Połączenia zarządzane przez klienta
- Niezarządzane połączenia z serwerem
- Niezarządzane połączenia klienckie

Więcej informacji na temat tych połączeń zawiera sekcja “Opcje połączenia WCF” na stronie 1390.

Interfejs inny niż SOAP/Non-JMS

Kanał niestandardowy produktu IBM MQ dla produktu WCF obsługuje zarówno interfejs SOAP/JMS (dostępny w produkcie IBM WebSphere MQ 7.0.1). i interfejs Non-SOAP/Non-JMS .

Architektura WCF jest przedstawiona na poniższym diagramie:



Rysunek 155. Architektura WCF dla interfejsu Non-SOAP/Non-JMS

Korzystanie z niestandardowych kanałów produktu IBM MQ dla produktu WCF

Przegląd informacji dostępnych dla programistów korzystających z niestandardowych kanałów produktu IBM MQ dla programu Windows Communication Foundation (WCF).

Program Microsoft Windows Communication Foundation stanowi podstawę dla usług Web Service i obsługi przesyłania komunikatów w środowisku Microsoft.NET Framework 3. Produkt IBM WebSphere MQ 7.0 lub nowszy może być używany jako kanał niestandardowy w systemie 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 przez produkt JMS w produkcie IBM WebSphere MQ 7.0 lub nowszym. Aplikacje mogą następnie komunikować się z usługami udostępnianą przez system WCF lub za pomocą infrastruktury usługi WebSphere SOAP over JMS .

Niestandardowe funkcje i możliwości kanału WCF

Poniższe tematy zawierają informacje na temat niestandardowych funkcji i możliwości kanału WCF.

Niestandardowe kształty kanałów WCF

Przegląd niestandardowych kształtów kanałów, które mogą być używane przez produkt IBM MQ , można znaleźć w kanałach niestandardowych produktu Microsoft Windows Communication Foundation (WCF).

Kanał niestandardowy produktu IBM MQ dla WCF obsługuje dwa kształty kanałów:

- Jednokierunkowa
- Żądanie-odpowiedź

WCF automatycznie wybiera kształt kanału zgodnie z udostępnianym kontraktem serwisowym.

Kontrakty zawierające metody, które używają tylko parametru **IsOneWay** , są obsługiwane przez jednokierunkowy kształt kanału, na przykład:

```
[OperationContract(IsOneWay = true)]
void printString(String text);
```

Kontrakty zawierające mieszaninę metod jednokierunkowych i żądanie-odpowiedź lub wszystkie metody żądanie-odpowiedź są obsługiwane przez kształt kanału odpowiedzi na żądanie. Na przykład:

```
    [OperationContract]
    int subtract(int a, int b);

    [OperationContract(IsOneWay = true)]
    void printString(string text);
```

Uwaga: Podczas mieszania metod jednokierunkowych i żądanie-odpowiedź w tej samej umowie należy upewnić się, że zachowanie jest zgodne z przeznaczeniem, szczególnie w przypadku pracy w środowisku mieszanym, ponieważ metody jednokierunkowe oczekują na odebranie odpowiedzi o wartości NULL z usługi.

Kanał jednokierunkowy

Niestandardowy kanał produktu 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

Niestandardowy kanał odpowiedzi żądanie-odpowiedź dla produktu IBM MQ używany jest na przykład w celu asynchronicznego wysyłania komunikatów w dwóch kierunkach; ta sama instancja klienta musi być używana do asynchronicznego przesyłania komunikatów. Kanał może wysyłać komunikaty w jednym kierunku, na przykład z menedżera kolejek klienta do kolejki w usłudze WCF, a następnie wysłać komunikat odpowiedzi z WCF do kolejki w menedżerze kolejek klienta.

Nazwy i wartości parametrów URI WCF

Nazwy i wartości parametrów URI dla interfejsu SOAP/JMS oraz interfejsu innego niż SOAP/Non JMS .

Interfejs SOAP/JMS

connectionFactory
Parametr connectionFactory jest wymagany.

InitialContextFactory
Parametr initialContextFactory jest wymagany i musi być ustawiony na wartość "com.ibm.mq.jms.Nojndi" w celu zapewnienia zgodności z produktem WebSphere Application Server i innymi produktami.

Interfejs inny niż SOAP/Non JMS

Format identyfikatora URI jest taki sam jak w przypadku specyfikacji MA93 . Więcej informacji na temat specyfikacji IRI IBM MQ można znaleźć w sekcji SupportPac - MA93 .

Składnia identyfikatora URI produktu 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 o nazwie example.com na porcie 1414 i umieścić komunikaty żądań trwałych do kolejki o nazwie SampleQ w menedżerze kolejek QM1. IRI określa, że dostawca usług będzie umieszczać 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 włączonym protokołem TLS

Aby utworzyć połączenia TLS przy użyciu klienta/usługi WCF, należy ustawić następujące właściwości odpowiednimi wartościami w identyfikatorze URI. Wszystkie właściwości, które są poprzedzane znakiem "*", są obowiązkowe w celu nawiązania połączenia zabezpieczonego.

- **sslKeyRepository**: *SYSTEM lub *USER
- * **sslCipherSpec**: poprawna wartość atrybutu 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"
```

Zapewniony kanał niestandardowy WCF

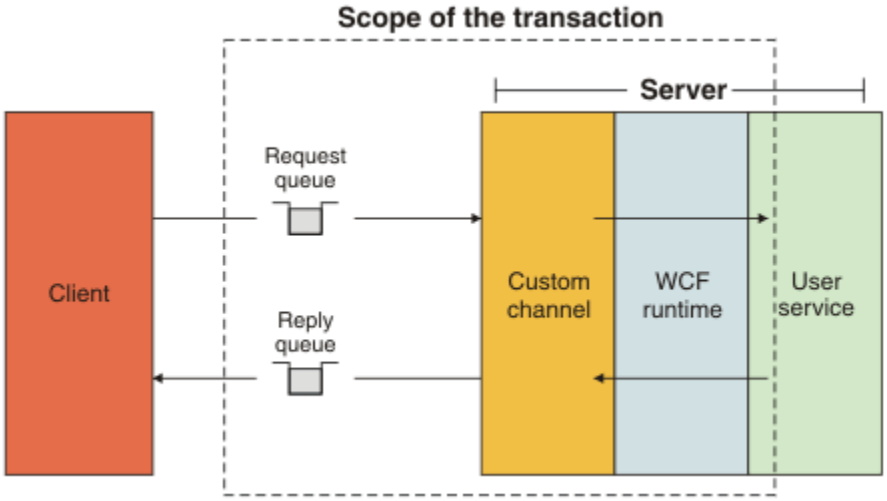
Zapewnione dostarczanie gwarantuje, że żądanie usługi lub odpowiedź są przydzielane i nie są tracone.

Odebrano komunikat z żądaniem, a każdy komunikat odpowiedzi jest wysyłany w lokalnym punkcie synchronizacji transakcji, który może zostać wycofany w przypadku niepowodzenia w czasie wykonywania. Przykłady tych awarii to: nieobsługiwany wyjątek zgłoszony przez usługę, niepowodzenie wysłania komunikatu do usługi lub niepowodzenie dostarczenia komunikatu odpowiedzi.

`AssuredDelivery` to zapewniony atrybut dostarczania, który można określić w umowie o świadczenie usług, aby zagwarantować, że wszystkie komunikaty żądania odebrane przez usługę oraz wszystkie komunikaty odpowiedzi wysłane z usługi nie zostaną utracone w przypadku niepowodzenia w czasie wykonywania.

Aby komunikaty były zachowywane również w przypadku awarii systemu lub wyłączenia zasilania, komunikaty muszą być wysyłane jako trwałe. Aby można było używać komunikatów trwałych, aplikacja kliencka musi mieć tę opcję określoną w identyfikatorze URI punktu końcowego.

Rozproszone transakcje nie są obsługiwane, a zasięg transakcji nie wykracza poza przetwarzanie komunikatów żądania i odpowiedzi wykonywane przez produkt IBM MQ. Każda praca wykonana w ramach usługi może zostać ponownie uruchomiona w wyniku awarii, która spowoduje ponowne odebraną wiadomość. Na poniższym diagramie przedstawiono zakres transakcji:



Zapewnione dostarczanie jest włączone 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 punktach:

- Jeśli kanał wykryje, 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 wysłany do usługi. Nieobsłużone wyjątki zgłaszane przez operację usługi są zawsze powtarzane aż do momentu, gdy komunikat zostanie ponownie dostarczony przez maksymalną liczbę razy określoną przez właściwość progu wycofania kolejki żądań. Więcej informacji na ten temat zawiera sekcja: “Niestandardowe komunikaty nieprzetwarzalne kanału WCF” na stronie 1388
- Kanał wykonuje odczytywanie, przetwarzanie i odpowiadanie na każdy komunikat żądania jako operację atomową przy użyciu pojedynczego wątku wykonania w celu wymuszenia integralności transakcyjnej. Aby umożliwić współbieżne uruchamianie operacji serwisowych, kanał WCF umożliwia tworzenie wielu instancji kanału. Liczba instancji kanału dostępnych do przetwarzania żądań jest sterowana przez właściwość powiązania `MaxConcurrentCalls`. Więcej informacji na ten temat zawiera sekcja: “Opcje konfiguracji powiązania WCF” na stronie 1396
- Zapewniona funkcja dostarczania korzysta zarówno z punktów rozszerzalności IOperationInvoker , jak i IErrorHandler WCF. Jeśli te punkty rozszerzalności są używane zewnętrznie przez aplikację, aplikacja musi upewnić się, że wszystkie zarejestrowane wcześniej punkty rozszerzalności są wywoływane. Niezastosowanie się do procedury IErrorHandler może spowodować, że błędy nie zostaną zgłoszone. Niepowodzenie operacji IOperationInvoker może spowodować, że WCF przestanie odpowiadać.

Niestandardowe zabezpieczenia kanału WCF

Niestandardowy kanał produktu IBM MQ dla systemu WCF obsługuje użycie protokołu TLS tylko dla niezarządzanych połączeń klientów z menedżerem kolejek.

Określ protokół TLS, używając pozycji w tabeli definicji kanału klienta (CCDT). Więcej informacji na temat CCDTs zawiera sekcja Tabela definicji kanału klienta.

Tabele definicji kanału klienta WCF (CCDT)

Niestandardowy kanał produktu IBM MQ dla systemu WCF obsługuje użycie tabel definicji kanału klienta (CCDT) w celu skonfigurowania informacji o połączeniu dla połączeń klienckich.

CCDTs są kontrolowane przez te dwie zmienne środowiskowe:

- *MQCHLLIB* określa katalog, w którym znajduje się tabela.
- *MQCHLTAB* określa nazwę pliku tabeli.

Jeśli te zmienne środowiskowe są zdefiniowane, mają one pierwszeństwo przed wszelkimi szczegółami połączenia klienta określonymi w identyfikatorze URI.

Więcej informacji na temat tabel definicji kanału klienta zawiera sekcja Tabela definicji kanału klienta.

Pojęcia pokrewne

Tabela definicji kanału klienta

Niestandardowe komunikaty nieprzetwarzalne kanału WCF

Gdy usługa nie może przetworzyć komunikatu żądania lub nie dostarczy komunikatu odpowiedzi do kolejki odpowiedzi, komunikat jest traktowany jako komunikat trucizny.

Komunikaty żądania trucizny

Jeśli komunikat żądania nie może zostać przetworzony, jest traktowany jako komunikat nieprzetwarzalny. To działanie uniemożliwia usłudze ponowne odbieranie tego samego komunikatu nieprzetwarzalnego. W przypadku nieprzetwarzalnego komunikatu żądania, który ma być traktowany jako komunikat nieprzetwarzalny, jedna z następujących sytuacji musi być prawdziwa:

- Liczba wycofań komunikatów przekroczyła próg wycofania określony w kolejce żądań, który ma miejsce tylko wtedy, gdy dla usługi została określona zapewniona dostawa. Więcej informacji na temat gwarantowanego dostarczania zawiera sekcja: “Zapewniony kanał niestandardowy WCF” na stronie 1386
- Komunikat nie został poprawnie sformatowany i nie mógł zostać zinterpretowany jako komunikat SOAP over JMS .

Komunikaty odpowiedzi trucizny

Jeśli usługa nie dostarczy komunikatu odpowiedzi do kolejki odpowiedzi, komunikat odpowiedzi jest traktowany jako komunikat nieprzetwarzalny. W przypadku komunikatów odpowiedzi to działanie umożliwia późniejsze odtworzenie komunikatów odpowiedzi w celu określenia problemu z pomocą.

Obsługa komunikatów trujących

Działanie podjęte dla komunikatu nieprzetwarzalnego zależy od konfiguracji menedżera kolejek oraz od wartości ustawionych w opcjach raportu komunikatu. W przypadku protokołu SOAP over JMSnastępujące opcje raportu są domyślnie ustawiane na komunikaty żądań i nie można ich konfigurować:

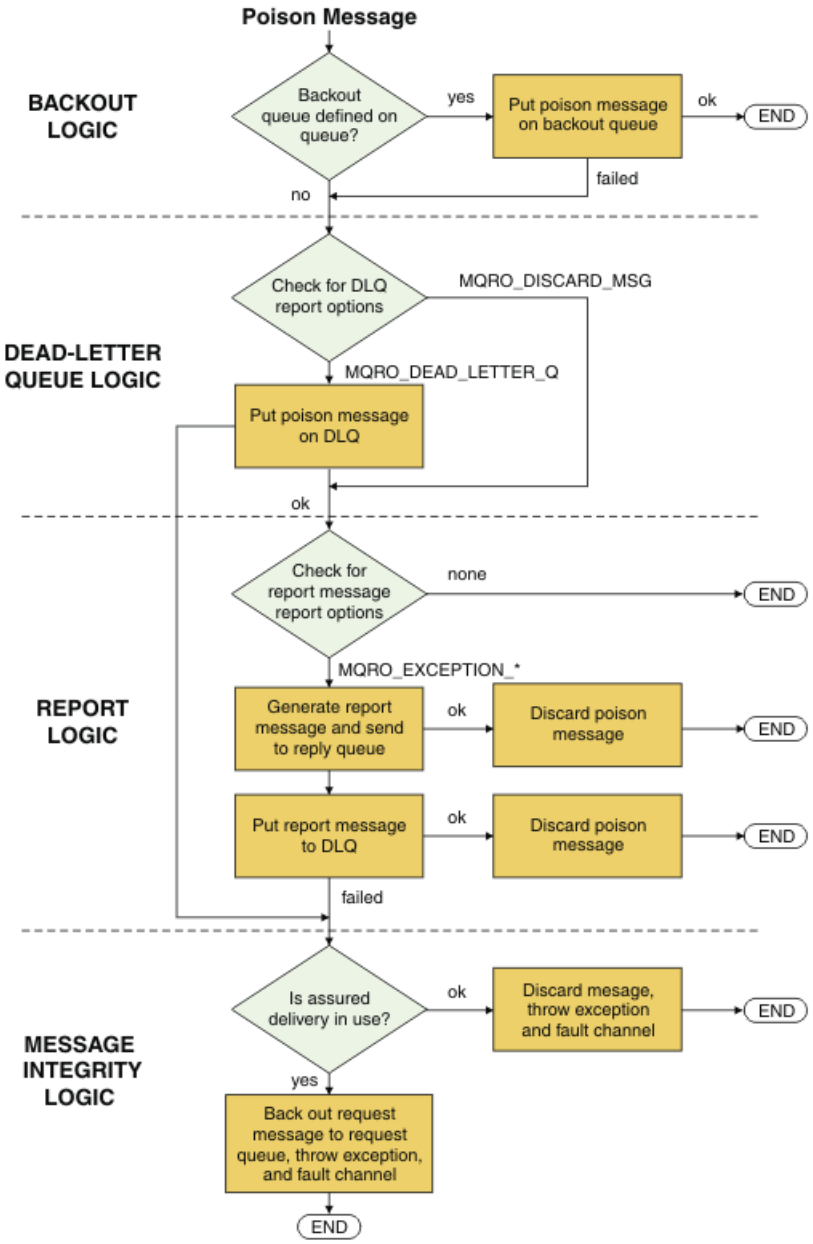
- MQRO_EXCEPTION_WITH_FULL_DATA
- MQRO_EXPIRATION_WITH_FULL_DATA
- MQRO_DISCARD_MSG

W przypadku protokołu SOAP over JMSnastępująca opcja raportu jest domyślnie ustawiana na komunikaty odpowiedzi i nie może być konfigurowalna:

- MQRO_DEAD_LETTER_Q

Jeśli komunikaty pochodzą ze źródła innego niż WCF, należy zapoznać się z dokumentacją dla tego źródła.

Na poniższym diagramie przedstawiono możliwe działania i kroki podjęte w przypadku niepowodzenia obsługi komunikatów nieprzetwarzalnych:



Możliwości komunikatów produktu IBM MQ dla aplikacji WCF

Inne niż SOAP/Non-JMS (to znaczy IBM MQ) możliwości komunikatów dla aplikacji WCF.

W przypadku interfejsu Non-SOAP/Non-JMS możliwości komunikatów produktu IBM MQ dla aplikacji WCF są następujące:

- Aplikacje WCF mogą wysyłać i odbierać podstawowe komunikaty produktu IBM MQ , które mogą być przetwarzane przez dowolną aplikację produktu IBM MQ .
- Aplikacje WCF mają pełną kontrolę nad aktualizowaniem deskryptora MQMD i ładunku.
- Klient WCF może wysyłać komunikaty produktu IBM MQ , które mogą być używane przez dowolnego klienta produktu IBM MQ , na przykład klientów C, Java, JMSi .NET .

Interfejs WCF for Non-SOAP/Non-JMS musi używać następujących klas w celu ustawienia ładunku komunikatu i deskryptora MQMD dla komunikatu:

- WmqStringKomunikat dla ładunku typu String
- Komunikat WmqBytesdla ładunku typu Bajty
- Komunikat WmqXmldla ładunku typu XML

Aby ustawić ładunek komunikatu, należy użyć właściwości **Data** w przypadku komunikatu WmqStringMessage, WmqBytesMessage lub WmqXmlMessage class, w zależności od typu ładunku. Na przykład, aby ustawić ładunek typu String, należy użyć następującego kodu:

```
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 nawiązywania połączenia niestandardowego kanału IBM MQ dla WCF z menedżerem kolejek. Należy wziąć pod uwagę, który typ połączenia najlepiej odpowiada wymaganiom użytkownika.

Więcej informacji na temat opcji połączenia zawiera sekcja: “Różnice między połączeniami” na stronie 587

Więcej informacji na temat architektury WCF można znaleźć pod adresem: “Architektura WCF” na stronie 1382

Niezarządzane połączenie klienta

Połączenie wykonane w tym trybie łączy się jako klient IBM MQ z serwerem IBM MQ działającym na komputerze lokalnym lub na komputerze zdalnym.

Aby użyć niestandardowego kanału produktu IBM MQ dla systemu WCF jako klienta IBM MQ , można go zainstalować przy użyciu produktu IBM MQ MQI clientna serwerze IBM MQ lub na osobnym komputerze.

Połączenie z serwerem niezarządzanym

W przypadku użycia w trybie powiązań serwera, kanał niestandardowy produktu IBM MQ dla WCF korzysta z interfejsu API menedżera kolejek, a nie komunikując się za pośrednictwem sieci. Użycie połączeń powiązań zapewnia lepszą wydajność aplikacji IBM MQ niż korzystanie z połączeń sieciowych.

Aby korzystać z połączenia powiązań, należy zainstalować niestandardowy kanał produktu IBM MQ dla produktu WCF na serwerze IBM MQ .

Połączenie z klientem zarządzanym

Połączenie wykonane w tym trybie łączy się jako klient IBM MQ z serwerem IBM MQ działającym na komputerze lokalnym lub na komputerze zdalnym.

Niestandardowe klasy kanału IBM MQ dla programu .NET 3 łączące się w tym trybie pozostają w kodzie zarządzanym .NET i nie wywołują żadnych połączeń z usługami rodzimymi. Więcej informacji na temat kodu zarządzanego można znaleźć w dokumentacji produktu Microsoft .

Istnieje wiele ograniczeń dotyczących korzystania z zarządzanego klienta. Więcej informacji na temat tych ograniczeń można znaleźć w sekcji “Połączenia zarządzane przez klienta” na stronie 587.

Tworzenie i konfigurowanie niestandardowego kanału produktu IBM MQ dla systemu WCF

Niestandardowe kanały produktu IBM MQ dla produktu WCF działają w taki sam sposób, jak kanały transportowe WCF oferowane przez produkt Microsoft. Kanał niestandardowy produktu IBM MQ dla WCF można utworzyć na jeden z dwóch sposobów.

O tym zadaniu

Kanał niestandardowy produktu IBM MQ integruje się z systemem WCF jako kanał transportowy WCF i jako taki musi być sparowany z koderem komunikatów i opcjonalnymi kanałami protokołu, dzięki czemu

może utworzyć kompletny stos kanałów, który może być używany przez aplikację. W celu pomyślnego utworzenia pełnego stosu kanału wymagane są dwa elementy:

1. Definicja powiązania: określa, które elementy są wymagane do zbudowania stosu kanału aplikacji, w tym kanału transportowego, kodera komunikatów i wszystkich protokołów, a także wszelkich ogólnych ustawień konfiguracyjnych. W przypadku kanału niestandardowego definicja powiązania musi zostać utworzona w postaci powiązania niestandardowego WCF.
2. Definicja punktu końcowego: łączy umowę o świadczenie usług z definicją powiązania, a także udostępnia rzeczywisty identyfikator URI połączenia, który opisuje miejsce, w którym aplikacja może nawiązać połączenie. W przypadku kanału niestandardowego identyfikator URI jest w postaci identyfikatora URI protokołu SOAP over 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 z poziomu kodu aplikacji.

Decyzja, nad którą metoda powinna zostać użyta do utworzenia definicji, musi być oparta na wymaganiach aplikacji w następujący sposób:

- Metoda administracyjna konfiguracji zapewnia elastyczność w zakresie zmiany szczegółów usługi i klienta po wdrożeniu bez odbudowowania aplikacji.
- Programowa metoda konfiguracji zapewnia większą ochronę przed błędami konfiguracji oraz możliwość dynamicznego generowania konfiguracji w czasie wykonywania.

Tworzenie niestandardowego kanału WCF administracyjnego przez dostarczanie informacji o powiązaniach i punktach końcowych w pliku konfiguracyjnym aplikacji

Kanał niestandardowy produktu IBM MQ dla WCF jest kanałem WCF na poziomie transportu. Aby można było używać kanału niestandardowego, należy zdefiniować punkt końcowy i powiązanie. Definicje te można wykonać, podając informacje o powiązaniach i punktach końcowych w pliku konfiguracyjnym aplikacji.

Aby skonfigurować i używać niestandardowego kanału produktu IBM MQ dla systemu WCF, który jest kanałem WCF na poziomie transportu, należy zdefiniować powiązanie i definicję punktu końcowego. Powiązanie przechowuje informacje konfiguracyjne dla kanału, a definicja punktu końcowego zawiera szczegóły połączenia. Definicje te mogą być tworzone na dwa sposoby:

- Programowo bezpośrednio z kodu aplikacji, zgodnie z opisem w tym miejscu: “Tworzenie kanału niestandardowego WCF przez programowe informacje o powiązaniach i punktach końcowych” na stronie 1393
- Administracyjnie, podając szczegółowe informacje w pliku konfiguracyjnym aplikacji, zgodnie z opisem w poniższej procedurze.

Plik konfiguracyjny klienta lub aplikacji usługi nosi nazwę *yourappname*`.exe.config` , gdzie *nazwa_aplikacje_użytkownika* jest nazwą aplikacji. Plik konfiguracyjny aplikacji jest najłatwiej modyfikowany za pomocą narzędzia Microsoft edytora konfiguracji usługi o nazwie `SvcConfigEditor.exe` w następujący sposób:

- Uruchom narzędzie edytora konfiguracji produktu `SvcConfigEditor.exe` . Domyślne miejsce instalacji dla tego narzędzia to: *Drive:*`\Program Files\Microsoft SDKs\Windows\v6.0\Bin\SvcConfigEditor.exe` , gdzie *Napęd:* jest nazwą napędu instalacyjnego.

Krok 1: Dodawanie rozszerzenia elementu binding w celu włączenia WCF w celu znalezienia kanału niestandardowego

1. Kliknij prawym przyciskiem myszy opcję **Zaawansowane** > **Rozszerzenie** > **element powiązania** , aby otworzyć menu, a następnie wybierz opcję **Nowy** .
2. Wypełnij pola zgodnie z poniższą tabelą:

Tabela 197. Nowe pola elementu powiązania	
Pole	**Wartość**
Nazwa	`IBM.XMS.WCF.SoapJmsIbmTransportChannel`
Typ	Przejdź do `IBM.XMS.WCF.dll` w globalnej pamięci podręcznej zespołu (Global Assembly Cache-GAC) i wybierz `IBM.XMS.WCFSoapJmsIbmTransportBindingElementConfig`

Krok 2: tworzenie niestandardowej definicji powiązania, która paruje kanał niestandardowy 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 poniższą tabelą:

Tabela 198. 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 opcję *IBM.XMS.WCF.SoapJmsIbmTransportChannel* Powiązanie transportu z powiązania, które zostało utworzone w: “Krok 2: tworzenie niestandardowej definicji powiązania, która paruje kanał niestandardowy z koderem komunikatów WCF” na stronie 1392
2. Wprowadź wszystkie wymagane zmiany w wartościach domyślnych właściwości zgodnie z opisem w: “Opcje konfiguracji powiązania WCF” na stronie 1396

Krok 4: Tworzenie definicji punktu końcowego

Utwórz definicję punktu końcowego, która odwołuje się do powiązania niestandardowego, które zostało utworzone w: “Krok 2: tworzenie niestandardowej definicji powiązania, która paruje kanał niestandardowy z koderem komunikatów WCF” na stronie 1392 , i udostępnia szczegóły połączenia usługi. Sposób określania tych informacji jest zależny 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 poniższą tabelą:

Tabela 199. Pola nowego punktu końcowego klienta	
Pole	**Wartość**
Nazwa	`Endpoint_WMQ`

Tabela 199. Pola nowego punktu końcowego klienta (kontynuacja)	
Pole	**Wartość**
Adres	*Identyfikator URI SOAP/JMS opisujący szczegóły połączenia WMQ wymagane w celu uzyskania dostępu do usługi.* Więcej informacji na ten temat zawiera sekcja "IBM MQ niestandardowy kanał dla formatu adresu URI punktu końcowego WCF" na stronie 1395 .
Łączy	`customBinding`
BindingConfiguration	`CustomBinding_WMQ`
Contract (kontrakt)	*Nazwa interfejsu umowy usługi*

W przypadku aplikacji usługi należy dodać 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 wypełniaj pola zgodnie z poniższą tabelą:

Tabela 200. Nowe pola punktu końcowego usługi	
Pole	**Wartość**
Nazwa	`Endpoint_WMQ`
Adres	*Identyfikator URI SOAP/JMS opisujący szczegóły połączenia WMQ wymagane w celu uzyskania dostępu do usługi.* Więcej informacji na ten temat zawiera sekcja "IBM MQ niestandardowy kanał dla formatu adresu URI punktu końcowego WCF" na stronie 1395 .
Łączy	`customBinding`
BindingConfiguration	`CustomBinding_WMQ`
Contract (kontrakt)	*Nazwa klasy implementacji usługi*

Tworzenie kanału niestandardowego WCF przez programowe informacje o powiązaniach i punktach końcowych

Kanał niestandardowy produktu IBM MQ dla WCF jest kanałem WCF na poziomie transportu. Aby można było używać kanału niestandardowego, należy zdefiniować punkt końcowy i powiązanie, a te definicje można wykonać programowo bezpośrednio z poziomu kodu aplikacji.

Aby skonfigurować i używać niestandardowego kanału produktu IBM MQ dla systemu WCF, który jest kanałem WCF na poziomie transportu, należy zdefiniować powiązanie i definicję punktu końcowego. Powiązanie przechowuje informacje konfiguracyjne dla kanału, a definicja punktu końcowego zawiera szczegóły połączenia. Więcej informacji na ten temat zawiera sekcja "Korzystanie z przykładów WCF" na stronie 1404.

Definicje te mogą być tworzone na dwa sposoby:

- Administracyjnie, udostępniając szczegóły w pliku konfiguracyjnym aplikacji, zgodnie z opisem w sekcji "Tworzenie niestandardowego kanału WCF administracyjnego przez dostarczanie informacji o powiązaniach i punktach końcowych w pliku konfiguracyjnym aplikacji" na stronie 1391.
- Programowo bezpośrednio z kodu aplikacji, zgodnie z opisem w poniższych podtematach.

Programowo definiowanie informacji o powiązaniu i punkcie końcowym: interfejs SOAP/JMS
W przypadku interfejsu SOAP/JMS można zdefiniować punkt końcowy i powiązanie programowo bezpośrednio z poziomu kodu aplikacji.

O tym zadaniu

Aby programowo dostarczyć informacje o powiązaniu i punkcie końcowym, należy dodać wymagany kod do aplikacji, wykonując następujące kroki.

Procedura

1. Utwórz instancję elementu powiązania transportu dla kanału, dodając do aplikacji następujący kod:

   ```
   SoapJmsIbmTransportBindingElement transportBindingElement = new
   SoapJmsIbmTransportBindingElement();
   ```

2. Ustaw wszystkie wymagane właściwości powiązania, na przykład poprzez dodanie do aplikacji następującego kodu w celu ustawienia `ClientConnectionMode`:

   ```
   transportBindingElement.ClientConnectionMode = XmsWCFBindingProperty.AS_URI;
   ```

3. Utwórz niestandardowe powiązanie, które paruje kanał transportowy za pomocą programu kodującego komunikaty, dodając następujący kod do aplikacji:

   ```
   Binding binding = new CustomBinding(new TextMessageEncodingBindingElement(),
   transportBindingElement);
   ```

4. Utwórz identyfikator URI SOAP/JMS .

   Identyfikator URI SOAP/JMS , który opisuje szczegóły połączenia IBM MQ wymagane do uzyskania dostępu do usługi, musi być podany jako adres punktu końcowego. Podany adres zależy od tego, czy kanał jest używany dla aplikacji usługowej, czy 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 usługowych 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 można znaleźć w sekcji “IBM MQ niestandardowy kanał dla formatu adresu URI punktu końcowego WCF” na stronie 1395.

Programowo definiowanie informacji o powiązaniu i punkcie końcowym: interfejs inny niż SOAP/Non-JMS
W przypadku interfejsu innego niż SOAP/Non-JMS można zdefiniować punkt końcowy i powiązanie programowo bezpośrednio z poziomu kodu aplikacji.

O tym zadaniu

Aby programowo dostarczyć informacje o powiązaniu i punkcie końcowym, należy dodać wymagany kod do aplikacji, wykonując następujące kroki.

Procedura

1. Utwórz element WmqBinding , dodając następujący kod do aplikacji:

   ```
   WmqBinding binding = new WmqBinding();
   ```

   Ten kod tworzy powiązanie, które pary elementów WmqMsgEncodingElement i WmqIbmTransportBindingwymagane dla interfejsu Non-SOAP/Non-JMS .
2. Podaj identyfikator URI wmq:, który opisuje szczegóły połączenia IBM MQ wymagane do uzyskania dostępu do usługi.

   Sposób udostępnienia identyfikatora URI wmq: // zależy od tego, czy kanał jest używany dla aplikacji usługowej, czy aplikacji klienckiej.

   - W przypadku aplikacji klienckich identyfikator URI wmq: // musi zostać utworzony jako element EndpointAddress w następujący sposób:

     ```
     EndpointAddress address = new EndpointAddress
     ("wmq://localhost:1414/msg/queue/Q1?connectQueueManager=QM1&replyTo=Q2");
     ```

   - W przypadku aplikacji usługowych 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");
     ```

IBM MQ niestandardowy kanał dla formatu adresu URI punktu końcowego WCF

Usługa Web Service jest określana przy użyciu identyfikatora URI (Universal Resource Identifier), który udostępnia położenie i szczegóły połączenia. Format identyfikatora URI zależy od tego, czy używany jest interfejs SOAP/JMS , czy też interfejs inny niż SOAP/Non-JMS .

Interfejs SOAP/JMS

Format identyfikatora URI, który jest obsługiwany w transporcie produktu IBM MQ dla protokołu SOAP, pozwala na uzyskanie kompleksowego poziomu kontroli nad parametrami i opcjami specyficznymi dla SOAP/ IBM MQ podczas uzyskiwania dostępu do usług docelowych. Ten format jest zgodny z produktem WebSphere Application Server i z produktem CICS, co ułatwia integrację produktu IBM MQ z 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 powtórzony dowolną liczbę razy z drugim, a kolejne wystąpienia poprzedzają znak ampersand (&).

W nazwach parametrów rozróżniana jest wielkość liter, podobnie jak nazwy obiektów IBM MQ . Jeśli dowolny parametr zostanie określony więcej niż jeden raz, końcowe wystąpienie parametru będzie miało wpływ na to, że aplikacje klienckie mogą przesłonić wartości parametrów, dołączając do identyfikatora URI. Jeśli zostaną uwzględnione dodatkowe nierozpoznane parametry, zostaną one zignorowane.

Jeśli identyfikator URI jest przechowywany w łańcuchu XML, należy reprezentować znak ampersand jako "&". Podobnie, jeśli identyfikator URI jest zakodowany w skrypcie, należy zwrócić uwagę na znaki zmiany znaczenia, takie jak **&** , które w przeciwnym razie zostałyby zinterpretowane przez powłokę.

Poniżej przedstawiono 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
```

Dostarczane są tylko wymagane parametry (opcja `targetService` jest wymagana tylko dla usług .NET), a opcja `connectionFactory` nie ma żadnych opcji.

W tym przykładzie `connectionFactory` znajduje się w tym przykładzie z wieloma opcjami:

```
jms:/queue?destination=myQ@myRQM&connectionFactory=connectQueueManager(myconnQM)
binding(client)clientChannel(myChannel)clientConnection(myConnection)
&initialContextFactory=com.ibm.mq.jms.Nojndi
```

W tym przykładzie przedstawiono również opcję `sslPeerName` klasy `connectionFactory` . Wartość parametru sslPeerNazwa zawiera 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 pozwala na uzyskanie kompleksowego poziomu kontroli nad parametrami i opcjami specyficznymi dla 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
```

Aplikacja IRI informuje requester usług, że może użyć połączenia powiązania klienta TCP IBM MQ 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. IRI określa, że dostawca usług umieszcza odpowiedzi w kolejce o nazwie INS.QUOTE.REPLY w menedżerze kolejek BRANCH452. Format identyfikatora URI jest określony w polu SupportPac MA93. Więcej informacji na temat specyfikacji IBM MQ IRI znajduje się w sekcji SupportPac MA93: IBM MQ -Definicja usługi .

Opcje konfiguracji powiązania WCF

Istnieją dwa sposoby zastosowania opcji konfiguracji do informacji o powiązaniach kanałów niestandardowych. Właściwości można ustawić administracyjnie albo ustawić je programowo.

Opcje konfiguracji powiązania można ustawić na jeden z dwóch różnych 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, aby można było określić właściwość podczas inicjowania powiązania niestandardowego.

Ustawianie właściwości powiązania administracyjnie

Ustawienia właściwości powiązania można określić w pliku konfiguracyjnym aplikacji, na przykład: `app.config`. Plik konfiguracyjny jest generowany przez program **svcutil**, tak jak przedstawiono to w poniższych przykładach.

Interfejs SOAP/JMS

```
<customBinding>
...
      <IBM.XMS.WCF.SoapJmsIbmTransportChannel maxBufferPoolSize="524288"
      maxMessageSize="4000000" clientConnectionMode="0" maxConcurrentCalls="16"/>
...
</customBinding>
```

Interfejs inny niż SOAP/Non-JMS

```
<customBinding>
      <IBM.WMQ.WCF.WmqMsgEncodingElement/>
      <IBM.WMQ.WCF.WmqIbmTransportChannel maxBufferPoolSize="524288"
      maxMessageSize="65536" clientConnectionMode="managedclient"/>
</customBinding>
```

Programowo ustawianie właściwości powiązania

Aby dodać właściwość powiązania WCF w celu określenia trybu połączenia klienckiego, należy zmodyfikować kod usługi, aby określić właściwość podczas inicjowania powiązania niestandardowego.

Użyj następującego przykładu, aby określić tryb niezarządzanego połączenia klienckiego:

```
SoapJmsIbmTransportBindingElement
transportBindingElement = new SoapJmsIbmTransportBindingElement();
transportBindingElement.ClientConnectionMode = XmsWCFBindingProperty.CLIENT_UNMANAGED;

Binding sampleBinding = new CustomBinding(new TextMessageEncodingBindingElement(),
                                                    transportBindingElement);
```

Właściwości powiązania WCF

Tabela 201. Wartości właściwości powiązania podczas ustawiania administracyjnie lub programowo

Nazwa właściwości	Aplikacja kliencka lub usługowa	Wartość administracyjna	Wartość programowa	Opis
maxBufferPoolSize	Obie	Od 0 do 64 bitowej liczby całkowitej	Od 0 do 64 bitowej liczby całkowitej	Określa maksymalną wielkość pamięci, która może być używana do przechowywania buforów komunikatów WCF dla instancji kanału.
maxMessage, Wielkość	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 użyta dla pojedynczego komunikatu WCF.

Tabela 201. Wartości właściwości powiązania podczas ustawiania administracyjnie lub programowo (kontynuacja)

Nazwa właściwości	Aplikacja kliencka lub usługowa	Wartość administracyjna	Wartość programowa	Opis
Tryb clientConnection	Obie	0 (wartość domyślna) 1	AS_URI (wartość domyślna) KLIENT_UNMANAGED	Określa tryb połączenia klienta w kanale transportowym. Wartość `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. Wartość 0 jest wartością domyślną, jeśli nie jest ustawiony żaden tryb połączenia klienckiego. 1 oznacza, że tryb połączenia klienta jest klientem niezarządzanym. Używany tylko wtedy, gdy używane jest połączenie klienta.
Wywołania MaxConcurrent	Klient	Zakres wartości to 0-2 147 483 647 16 to wartość domyślna	Zakres wartości to 0-2 147 483 647 16 to wartość domyślna	Ta właściwość definiuje maksymalną liczbę współbieżnych operacji, które mogą być wykonywane na pojedynczym serwerze proxy klienta w dowolnym momencie. W przypadku uruchomienia większej liczby operacji są one umieszczane w kolejce do momentu zakończenia lub zakończenia operacji w toku. To ustawienie może być używane do sterowania maksymalną liczbę wątków i zasobów, które mogą być wykorzystywane przez pojedynczego proxy. `0` usuwa ten limit, umożliwiając jednoczesną próbę wykonania wszystkich operacji.

Tabela 201. Wartości właściwości powiązania podczas ustawiania administracyjnie lub programowo (kontynuacja)

Nazwa właściwości	Aplikacja kliencka lub usługowa	Wartość administracyjna	Wartość programowa	Opis
Wywołania MaxConcurrent	Usługa	Zakres ten wynosi 1-2 147 483 647 16 to wartość domyślna	Zakres ten wynosi 1-2 147 483 647 16 to wartość domyślna	Ta właściwość jest używana tylko wtedy, gdy funkcja gwarantowanego dostarczania jest włączona (więcej informacji na temat gwarantowanego dostarczania zawiera sekcja “Zapewniony kanał niestandardowy WCF” na stronie 1386). Określa on maksymalną liczbę współbieżnych operacji, które mogą być jednocześnie w toku dla danego punktu końcowego. Podczas zmiany tego ustawienia należy zachować ostrożność. Każda operacja współbieżna wymaga dodatkowych zasobów, w szczególności nowej instancji kanału niestandardowego i powiązanych wątków z puli wątków w celu działania żądań. Nadmierna alokacja może być bardzo wydajna i poważnie wpływa na wydajność. W celu obsługi tej właściwości należy utworzyć odpowiednią konfigurację puli wątków.

Usługi budowlane i usługi serwerowe dla WCF

Przegląd usług produktu Microsoft Windows Communication Foundation (WCF) wyjaśniających, w jaki sposób można tworzyć i konfigurować usługi WCF.

Kanał niestandardowy produktu IBM MQ dla usług WCF i WCF, które korzystają z tego kanału, może być udostępniany za pomocą następujących metod:

- Self-hosting
- Usługa Windows

Niestandardowy kanał produktu IBM MQ dla systemu WCF nie może być udostępniany w usłudze aktywacji procesów produktu Windows .

W poniższych tematach przedstawiono kilka prostych przykładów samodzielnego udostępniania, aby zademonstrować kroki związane z tym tematem. Dokumentacja elektroniczna produktu Microsoft WCF, zawierająca dalsze informacje oraz najnowsze informacje szczegółowe, znajduje się na stronie WWW MSDN produktu Microsoft pod adresem https://msdn.microsoft.com.

Budowanie aplikacji usług WCF przy użyciu metody 1: Samodzielne udostępnianie administracyjnie przy użyciu pliku konfiguracyjnego aplikacji

Po utworzeniu pliku konfiguracyjnego aplikacji otwórz instancję usługi i dodaj określony kod do aplikacji.

Zanim rozpoczniesz

Utwórz lub dokonaj edycji pliku konfiguracyjnego aplikacji dla usługi zgodnie z opisem w: "Tworzenie niestandardowego kanału WCF administracyjnego przez dostarczanie informacji o powiązaniach i punktach końcowych w pliku konfiguracyjnym aplikacji" na stronie 1391

O tym zadaniu

1. Utwórz instancję i otwórz instancję usługi na 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: Samodzielność programowo bezpośrednio z aplikacji

Dodaj właściwości powiązania, utwórz host usługi z instancją wymaganej klasy usługi i otwórz usługę.

Zanim rozpoczniesz

1. Dodaj odwołanie do niestandardowego pliku kanału `IBM.XMS.WCF.dll` do projektu. `IBM.XMS.WCF.dll` znajduje się w katalogu *`WMQInstallDir`*`\bin` , gdzie *WMQInstallDir* to katalog, w którym jest zainstalowany 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 zgodnie z opisem w: "Tworzenie kanału niestandardowego WCF przez programowe informacje o powiązaniach i punktach końcowych" na stronie 1393

O tym zadaniu

Jeśli wymagane są zmiany właściwości powiązania kanału, wykonaj następujące kroki:

1. Dodaj właściwości powiązania do produktu `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 produktu IBM MQ dla systemu WCF.

O tym zadaniu

Jeśli metadane usług muszą być ujawnione (aby narzędzia, takie jak produkt `svcutil` , mogły uzyskać do niego dostęp bezpośrednio z działającej usługi, a nie z pliku WSDL działającego w trybie bez połączenia), należy to zrobić, ujawniając metadane usług za pomocą punktu końcowego HTTP. Aby dodać ten dodatkowy punkt końcowy, można użyć następujących kroków.

1. Dodaj adres bazowy, w którym metadane muszą być ujawnione na hoście ServiceHost, na przykład:

```
ServiceHost service = new ServiceHost(typeof(TestService),
                      new Uri("http://localhost:8000/MyService"));
```

2. Dodaj następujący kod do obiektu ServiceHost przed otwarniem 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 WCF

Przegląd generowania i budowania aplikacji klienckich produktu Microsoft Windows Communication Foundation (WCF).

Aplikacja kliencka może zostać utworzona dla usługi WCF. Aplikacje klienckie są zwykle generowane przy użyciu narzędzia Microsoft ServiceModel Metadata Utility Tool (Svcutil.exe) w celu utworzenia wymaganych plików konfiguracyjnych i proxy, które mogą być używane bezpośrednio przez aplikację.

Generowanie proxy klienta WCF i plików konfiguracyjnych aplikacji przy użyciu narzędzia svcutil z metadanymi pochodzącym z uruchomionej 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 systemu WCF.

Zanim rozpoczniesz

Istnieją trzy wymagania wstępne dotyczące korzystania z narzędzia svcutil do tworzenia wymaganych plików konfiguracyjnych i 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 niestandardowych odwołań do punktów końcowych kanału produktu IBM MQ w celu wygenerowania klienta bezpośrednio z uruchomionej usługi.
- Kanał niestandardowy musi być zarejestrowany w danych konfiguracji dla usługi svcutil.

O tym zadaniu

W poniższych krokach wyjaśniono sposób generowania klienta dla usługi, która jest skonfigurowana do używania kanału niestandardowego produktu IBM MQ , ale również prezentuje metadane w czasie wykonywania za pośrednictwem osobnego portu HTTP:

1. Uruchom usługę WCF (usługa musi być uruchomiona przed uruchomieniem narzędzia svcutil).
2. Dodaj szczegóły z pliku konfiguracyjnego produktu `svcutil.exe` z katalogu głównego instalacji do aktywnego pliku konfiguracyjnego svcutil, zwykle produkt `C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\svcutil.exe.config` tak, aby program svcutil rozpoznał kanał niestandardowy produktu IBM MQ .
3. Uruchom komendę 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 Visual Studio Microsoft .

Co dalej

Jeśli nie można bezpośrednio pobrać metadanych usług, można użyć usługi svcutil w celu wygenerowania plików klienta z pliku wsdl. Więcej informacji na ten temat zawiera sekcja “Generowanie proxy klienta WCF i plików konfiguracyjnych aplikacji przy użyciu narzędzia svcutil przy użyciu pliku WSDL” na stronie 1402 .

Generowanie proxy klienta WCF i plików konfiguracyjnych aplikacji przy użyciu narzędzia svcutil przy użyciu pliku WSDL

Instrukcje dotyczące generowania klientów WCF na podstawie pliku WSDL, jeśli metadane usługi są niedostępne.

Jeśli nie można bezpośrednio pobrać metadanych usługi w celu wygenerowania klienta na podstawie metadanych z działającej usługi, można użyć usługi svcutil w celu wygenerowania plików klienta z pliku WSDL. Aby można było określić, że kanał niestandardowy produktu IBM MQ ma być używany, należy wprowadzić następujące modyfikacje w pliku WSDL:

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 w taki sposób, aby odwoływała się do nowej sekcji strategii, a następnie usuń dowolną definicję `transport` z bazowego elementu powiązania:

```
<wsdl:definitions ...>

        <wsdl:binding ...>
                <wsp:PolicyReference URI="#CustomerBinding_IWMQSampleContract_policy" />
                <[soap]:binding ... transport="" />
                ...
        </wsdl:binding>
</wsdl:definitions>
```

3. Uruchom komendę 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 proxy klienta z plikiem konfiguracyjnym aplikacji

Zanim rozpoczniesz

Utwórz lub edytuj plik konfiguracyjny aplikacji dla klienta zgodnie z opisem w: "Tworzenie niestandardowego kanału WCF administracyjnego przez dostarczanie informacji o powiązaniach i punktach końcowych w pliku konfiguracyjnym aplikacji" na stronie 1391

O tym zadaniu

Utwórz instancję serwera proxy klienta i otwórz instancję serwera proxy klienta. Parametr przekazany 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 odwołanie do niestandardowego pliku kanału `IBM.XMS.WCF.dll` do projektu. Katalog `IBM.XMS.WCF.dll` znajduje się w katalogu *WMQInstallDir*`\bin` , gdzie *WMQInstallDir* to katalog, w którym jest zainstalowany produkt IBM MQ .
2. Dodaj instrukcję *using* do przestrzeni nazw `IBM.XMS.WCF` , na przykład: `using IBM.XMS.WCF`
3. Utwórz instancję elementu wiążącego i punktu końcowego kanału zgodnie z opisem w: "Tworzenie kanału niestandardowego WCF przez programowe informacje o powiązaniach i punktach końcowych" na stronie 1393

O tym zadaniu

Jeśli wymagane są zmiany właściwości powiązania kanału, wykonaj następujące kroki.

1. Dodaj właściwości powiązania do składnika `transportBindingElement` , tak 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 w sposób przedstawiony na poniższej ilustracji, gdzie *powiązanie* i *adres punktu końcowego* są powiązaniem i adresem punktu końcowego skonfigurowanym w kroku 1 i przekazanego w:

```
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 produktu IBM MQ .

Aby zbudować przykładowe projekty, wymagany jest pakiet Microsoft.NET 3.5 SDK lub Microsoft Visual Studio 2008.

Przykład prostego klienta jednokierunkowego i serwera WCF

Ten przykład demonstruje niestandardowy kanał produktu IBM MQ używany do uruchamiania usługi WCF (Windows Communication Foundation) 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 proxy klienta WCF i plików konfiguracyjnych aplikacji przy użyciu narzędzia svcutil z metadanymi pochodzącym z uruchomionej usługi" na stronie 1401 .

Przykład został skonfigurowany z określonymi nazwami 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` , a następnie w aplikacji usługi 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 produktu JMS zawiera sekcja *IBM MQ Transport for 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 program nasłuchujący oczekiwał na komunikaty: Uruchom plik *MQ_INSTALLATION_PATH*

\tools\dotnet\samples\cs\wcf\samples\WCF\oneway\service\bin\Release\TestService.exe , gdzie *MQ_INSTALLATION_PATH* jest katalogiem instalacyjnym produktu 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 produktu IBM MQ.

   Aplikacja kliencka pętle pięciokrotnie wysyłając pięć komunikatów do *SampleQ*

Wyniki

Aplikacja usługi pobiera komunikaty z katalogu *SampleQ* i wyświetla `Hello World` na ekranie pięciokrotnie.

Co dalej

Przykład prostego klienta odpowiedzi żądania i serwera WCF

Ten przykład demonstruje niestandardowy kanał produktu IBM MQ używany do uruchamiania usługi WCF (Windows Communication Foundation) z klienta WCF przy użyciu kształtu kanału odpowiedzi na żądanie.

O tym zadaniu

Ta usługa udostępnia kilka prostych metod kalkulatora w celu dodania i odjęcia dwóch liczb, a następnie zwrócenia wyniku. Klient został wygenerowany przy użyciu narzędzia `svcutil` w celu pobrania metadanych usługi z oddzielnie ujawnionego punktu końcowego HTTP zgodnie z opisem w sekcji "Generowanie proxy klienta WCF i plików konfiguracyjnych aplikacji przy użyciu narzędzia svcutil z metadanymi pochodzącym z uruchomionej usługi" na stronie 1401 .

Przykład został skonfigurowany z określonymi nazwami zasobów w sposób opisany 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 , a następnie w aplikacji usługi 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 produktu JMS zawiera sekcja *IBM MQ Transport for 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 program nasłuchujący oczekiwał na komunikaty: Uruchom plik *MQ_INSTALLATION_PATH* \Tools\wcf\samples\WCF\requestreply\service\bin\Release\SimpleRequestReply_Service.exe , gdzie *MQ_INSTALLATION_PATH* jest katalogiem instalacyjnym produktu 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 produktu IBM MQ.

Wyniki

Po uruchomieniu klienta następujący proces jest uruchamiany i powtarzany cztery razy, więc w sumie wysyłane są pięć komunikatów:

1. Klient umieszcza komunikat żądania w pliku *SampleQ* i oczekuje na odpowiedź.
2. Usługa pobiera komunikat żądania z katalogu *SampleQ*.

3. Usługa dodaje i odejmuje niektóre wartości, korzystając z treści komunikatu.
4. Następnie usługa umieszcza wyniki w komunikacie w kolejce *SampleReplyQ*i oczekuje na umieszczenie nowego komunikatu przez klienta.
5. Klient pobiera komunikat z kolejki *SampleReplyQ* i wyświetla wyniki na ekranie.

Co dalej

Klient WCF do usługi .NET udostępnianej przez produkt IBM MQ

Przykładowe aplikacje klienckie i przykładowe aplikacje proxy usług są dostarczane zarówno dla produktów .NET , jak i dla produktu Java. Przykłady są oparte na usłudze wyceny papierów wartościowych, która pobiera żądanie dotyczące wyceny akcji, a następnie udostępnia wycenę akcji.

Zanim rozpoczniesz

The sample requires that the .NET SOAP over JMS service hosting environment is correctly installed and configured in IBM MQ and is accessible from a local queue manager.

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 produktu IBM MQ , na przykład: `C:\Program Files\IBM\MQ`
2. Upewnij się, że kompilator Java `javac` jest dostępny i znajduje się na ścieżce PATH.
3. Skopiuj plik `axis.jar` z katalogu `prereqs/axis` instalacyjnego dysku CD WebSphere 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* oznacza katalog, w którym jest zainstalowany produkt IBM MQ , na przykład: `C:\Program Files\IBM\MQ`
5. Upewnij się, że położenie produktu .NET zostało poprawnie określone w *MQ_INSTALLATION_PATH*`\bin\amqwcallWSDL.cmd` , gdzie *MQ_INSTALLATION_PATH* oznacza katalog, w którym zainstalowano produkt IBM MQ , na przykład: `C:\Program Files\IBM\MQ`. Położenie produktu .NET może być określone na przykład: `set msfwdir=%ProgramFiles% \Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin`

Gdy poprzednie kroki są kompletne, przetestuj i uruchom usługę:

1. Przejdź do katalogu roboczego SOAP over JMS .
2. Wprowadź jedną z następujących komend, aby uruchomić test weryfikujący i pozostaw uruchomiony program nasłuchujący usługi:
   - W systemie .NET: *MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt dotnet hold` , gdzie *MQ_INSTALLATION_PATH* oznacza katalog, w którym zainstalowano produkt IBM MQ .
   - W przypadku produktu AXIS: *MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt Dotnet2AxisClient hold` , gdzie *MQ_INSTALLATION_PATH* oznacza katalog, w którym zainstalowany jest produkt IBM MQ .

Argument `hold` powoduje, że obiekty nasłuchiwania są uruchamiane po zakończeniu testu.

Jeśli podczas tej konfiguracji zostaną zgłoszone błędy, można usunąć wszystkie zmiany, tak aby procedura mogła zostać zrestartowana w następujący sposób:

1. Usuń wygenerowany katalog SOAP over JMS .
2. Usuń menedżer kolejek.

O tym zadaniu

Ten przykład demonstruje połączenie klienta WCF z przykładową usługą .NET SOAP korzystającym z programu JMS udostępnionego w produkcie IBM MQ przy użyciu jednokierunkowego kształtu kanału. Usługa implementuje prosty przykład StockQuote , który powoduje wyjście z łańcucha tekstowego do konsoli.

Klient został wygenerowany przy użyciu pliku WSDL w celu wygenerowania plików klienta zgodnie z opisem w sekcji "Generowanie proxy klienta WCF i plików konfiguracyjnych aplikacji przy użyciu narzędzia svcutil przy użyciu pliku WSDL" na stronie 1402

Przykład został skonfigurowany z określonymi nazwami 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` , a następnie w aplikacji usługi w pliku *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WMQNET\default\service\WmqDefaultSample_StockQuoteDotNet.wsdl` , gdzie *MQ_INSTALLATION_PATH* oznacza katalog instalacyjny produktu IBM MQ. Więcej informacji na temat formatowania identyfikatora URI punktu końcowego produktu JMS zawiera sekcja *IBM MQ Transport for 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 produktu IBM MQ.

Aplikacja kliencka pęka pięć razy, wysyłając pięć komunikatów do kolejki próbkowania.

Wyniki

Aplikacja usługi pobiera komunikaty z kolejki przykładowej i wyświetla pięć razy `Hello World` ekranie pięć razy.

Klient WCF do usługi Axis Java obsługiwanej przez przykładowy program IBM MQ

Przykładowe aplikacje klienckie i przykładowe aplikacje proxy usług są dostarczane zarówno dla produktów Java , jak i dla produktu .NET. Przykłady są oparte na usłudze wyceny papierów wartościowych, która pobiera żądanie dotyczące wyceny akcji, a następnie udostępnia wycenę akcji.

Zanim rozpoczniesz

W tym przykładzie wymagane jest, aby środowisko usług serwerowych .NET SOAP over JMS zostało poprawnie zainstalowane i skonfigurowane w produkcie IBM MQ i było dostępne z 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 produktu IBM MQ , na przykład: `C:\Program Files\IBM\MQ`
2. Upewnij się, że kompilator Java `javac` jest dostępny i znajduje się na ścieżce PATH.
3. Skopiuj plik `axis.jar` z katalogu `prereqs/axis` instalacyjnego dysku CD WebSphere do katalogu instalacyjnego produktu IBM MQ .
4. Dodaj do zmiennej PATH: *MQ_INSTALLATION_PATH*`\Java\lib` , gdzie *MQ_INSTALLATION_PATH* oznacza katalog, w którym jest zainstalowany produkt IBM MQ , na przykład: `C:\Program Files\IBM\MQ`
5. Upewnij się, że położenie produktu .NET zostało poprawnie określone w *MQ_INSTALLATION_PATH*`\bin\amqwcallWSDL.cmd` , gdzie *MQ_INSTALLATION_PATH* oznacza

katalog, w którym zainstalowano produkt IBM MQ , na przykład: `C:\Program Files\IBM\MQ`. Położenie produktu .NET może być określone na przykład: `set msfwdir=%ProgramFiles% \Microsoft Visual Studio .NET 2003\SDK\v1.1\Bin`

Gdy poprzednie kroki są kompletne, przetestuj i uruchom usługę:

1. Przejdź do katalogu roboczego SOAP over JMS .
2. Wprowadź jedną z następujących komend, aby uruchomić test weryfikujący i pozostaw uruchomiony program nasłuchujący usługi:
   - W systemie .NET: *MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt dotnet hold` , gdzie *MQ_INSTALLATION_PATH* oznacza katalog, w którym zainstalowano produkt IBM MQ .
   - W przypadku produktu AXIS: *MQ_INSTALLATION_PATH*`\Tools\soap\samples\runivt Dotnet2AxisClient hold` , gdzie *MQ_INSTALLATION_PATH* oznacza katalog, w którym zainstalowany jest produkt IBM MQ .

Argument `hold` powoduje, że obiekty nasłuchiwania są uruchamiane po zakończeniu testu.

Jeśli podczas tej konfiguracji zostaną zgłoszone błędy, można usunąć wszystkie zmiany, tak 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 usługą przykładową Axis Java SOAP korzystającym z produktu JMS udostępnionym w produkcie IBM MQ przy użyciu jednokierunkowego kształtu kanału. Usługa implementuje prosty przykład StockQuote , który wyprowadza łańcuch tekstowy do pliku, który jest zapisywany w bieżącym katalogu.

Klient został wygenerowany przy użyciu pliku WSDL w celu wygenerowania plików klienta zgodnie z opisem w sekcji “Generowanie proxy klienta WCF i plików konfiguracyjnych aplikacji przy użyciu narzędzia svcutil przy użyciu pliku WSDL” na stronie 1402

Próbka została skonfigurowana z określonymi nazwami zasobów, zgodnie z opisem w niniejszym 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` , a także w aplikacji usługi w pliku *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WMQAxis\default\service\WmqDefaultSample_StockQuoteDotNet.wsdl` , gdzie *MQ_INSTALLATION_PATH* oznacza katalog instalacyjny dla 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 produktu IBM MQ.

Aplikacja kliencka pęka pięć razy, wysyłając pięć komunikatów do kolejki próbkowania.

Wyniki

Aplikacja usługi pobiera komunikaty z kolejki przykładowej i dodaje `Hello World` razy do pliku w bieżącym katalogu.

Klient WCF do usługi Java udostępnianej przez produkt WebSphere Application Server

Przykładowe aplikacje klienckie i przykładowe aplikacje proxy usługi są dostarczane dla produktu WebSphere Application Server 6. Udostępniana jest również usługa żądanie-odpowiedź.

Zanim rozpoczniesz

W tym przykładzie wymagane jest, aby używana była następująca konfiguracja produktu IBM MQ :

Tabela 202. Wymagana konfiguracja produktu IBM MQ	
Obiekt	**Wymagana nazwa**
Menedżer kolejek	QM1
Kolejka lokalna	HelloWorld
Kolejka lokalna	HelloWorldOdpowiedz

Ten przykład wymaga również, aby środowisko usług serwerowych WebSphere Application Server 6 zostało poprawnie zainstalowane i skonfigurowane. Produkt WebSphere Application Server 6 domyślnie używa połączenia z trybem powiązań do nawiązywania 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 programie WebSphere Application Server z następującą konfiguracją:
   - Ustaw nazwę JNDI fabryki połączeń na wartość `jms/HelloWorldQCF` .
   - Ustaw nazwę JNDI miejsca docelowego na `jms/HelloWorld`
3. Zainstaluj aplikację usługi Web Service `HelloWorldEJBEAR.ear` na serwerze aplikacji WebSphere 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 produktu IBM MQ.
   c. Nie należy zmieniać żadnej z opcji domyślnych w kreatorze i restartować serwer aplikacji po zainstalowaniu aplikacji.

Po zakończeniu konfigurowania serwera WAS przetestuj usługę, uruchamiając ją jeden raz:

1. Przejdź do katalogu roboczego Soap over JMS .
2. Wprowadź tę komendę, aby uruchomić przykład: *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WAS\TestClient.exe` , gdzie *MQ_INSTALLATION_PATH* jest katalogiem instalacyjnym produktu IBM MQ.

O tym zadaniu

Przykład demonstruje połączenie klienta WCF z przykładową usługą WebSphere Application Server SOAP korzystającym z produktu JMS dostarczonym w próbkach WCF dołączonych do produktu IBM MQprzy

użyciu kształtu kanału żądanie-odpowiedź. Komunikaty przepływa między WCF i WebSphere Application Server przy użyciu kolejek produktu IBM MQ . Usługa implementuje metodę `HelloWorld(...)` , która pobiera łańcuch i zwraca pozdrowienie 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 proxy klienta WCF i plików konfiguracyjnych aplikacji przy użyciu narzędzia svcutil z metadanymi pochodzącym z uruchomionej usługi" na stronie 1401 .

Przykład został skonfigurowany z określonymi nazwami 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` , a w aplikacji usługowej w *MQ_INSTALLATION_PATH* `\tools\wcf\samples\WAS\HelloWorldsEJBEAR.ear` , gdzie *MQ_INSTALLATION_PATH* jest katalogiem instalacyjnym produktu IBM MQ.

The service and client are based upon the service and client outlined in the IBM Developer article *Budowanie usługi Web Service produktu JMS przy użyciu protokołu SOAP przez produkty JMS i WebSphere Studio*. Więcej informacji na temat projektowania usług Web Service SOAP przez JMS zgodnych z kanałem niestandardowym produktu IBM MQ WCF zawiera serwis WWW 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 produktu IBM MQ.

Aplikacja kliencka uruchamia jednocześnie obie metody usługi, wysyłając dwa komunikaty do kolejki przykładowej.

Wyniki

Aplikacja usługi pobiera komunikaty z kolejki przykładowej i udostępnia odpowiedź na wywołanie metody `HelloWorld(...)` , które jest wysyłane przez aplikację kliencką 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. Używanie tego dokumentu nie daje żadnych praw do tych 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 z oprogramowaniem, Dział 49XA
3605 Highway 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 dotyczące interfejsu programistycznego, o ile są udostępniane, mają być pomocne podczas tworzenia oprogramowania aplikacji do użytku z tym programem.

Ten podręcznik zawiera informacje na temat planowanych 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 na temat diagnostyki, modyfikacji i strojenia nie należy używać jako interfejsu programistycznego, ponieważ może on ulec zmianie.

Znaki towarowe

IBM, logo IBM , ibm.com, są znakami towarowymi IBM Corporation, zarejestrowanymi w wielu systemach prawnych na całym świecie. Aktualna lista znaków towarowych IBM jest dostępna w serwisie WWW, 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 Microsoft Corporation w Stanach Zjednoczonych i/lub w 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 (http://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: