HITACHI
Inspire the Next

OpenTP1 Version 7
OpenTP1 メッセージキューイング機能

TP1/Message Queue プログラム作成の手引

手引書

3000-3-D92-40

前書き

■ 対象製品

マニュアル「TP1/Message Queue 使用の手引」を参照してください。

■ 輸出時の注意

本製品を輸出される場合には、外国為替及び外国貿易法の規制並びに米国輸出管理規則など外国の輸出関連法規をご確認の上、必要な手続きをお取りください。

なお、不明な場合は、弊社担当営業にお問い合わせください。

■ 商標類

■ 発行

2024 年 1 月　3000-3-D92-40

■ 著作権

変更内容

変更内容（3000-3-D92-40）TP1/Message Queue 07-60

追加・変更内容	変更個所
記載内容は変更なし。	−

単なる誤字・脱字などはお断りなく訂正しました。

変更内容（3000-3-D92-30）TP1/Message Queue 07-53

追加・変更内容
記載内容は変更なし。

変更内容（3000-3-D92-21）TP1/Message Queue 07-52

追加・変更内容
対象製品の記載を，マニュアル「TP1/Message Queue 使用の手引」を参照する内容に変更した。

変更内容（3000-3-D92-20）TP1/Message Queue 07-52

追加・変更内容
記載内容は変更なし。

変更内容（3000-3-D92-10）TP1/Message Queue 07-50

追加・変更内容
記載内容は変更なし。

はじめに

このマニュアルは，TP1/Message Queue を使用するアプリケーションの設計方法と作成方法について説明したものです。

TP1/Message Queue およびマニュアルは，米国 International Business Machines Corporation とのライセンス契約に基づき，IBM MQ（旧称：MQSeries）の MQI，MQFAP，MQ クラスタの仕様をベースに実装しています。

■ 対象読者

TP1/Message Queue を使用するアプリケーションプログラマを対象としています。

オペレーティングシステム，オンラインシステム，使うマシンの操作，およびアプリケーションプログラムのコーディングに使う高級言語（C 言語，C++言語，または COBOL 言語）の文法の知識があることを前提としています。

このマニュアルの記述は，マニュアル「OpenTP1 解説」の知識があることを前提としていますので，あらかじめお読みいただくことをお勧めします。

また，次のマニュアルを理解されていることを前提としています。

- TP1/Message Queue 使用の手引（3000-3-D90）

■ その他の前提条件

このマニュアルをお読みになる際のその他の前提情報については，マニュアル「TP1/Message Queue 使用の手引」を参照してください。

目次

図目次

表目次

1 概要

この章では，メッセージキューイングの概要について説明します。

1.1 メッセージキューイングの概要

メッセージキューイングは長年にわたりデータ処理に使用されてきました。近年では電子メールでよく使用されています。キューイングなしで電子メッセージを送信するには，経路にある各ノードについて，メッセージを転送可能な状態にあることやアドレスが記載されていることを意識しなければなりません。キューイングシステムの場合，システムがメッセージを転送できるようになるまで，メッセージは中間ノードに保存されます。最終目的地に到着すると，あて先が読み込み可能になるまで，メッセージは電子メール箱に保存されます。

しかし，多くの複雑なビジネストランザクションが今日もキューイングなしで処理されています。巨大なネットワークでは，システムは何千ものコネクションを使用可能な状態で維持することになります。このためシステムの一部に障害が発生すると，システムの大部分が使用不能になります。

メッセージキューイングはプログラム用の電子メールであると考えることができます。メッセージキューイング環境では，各アプリケーションは特定の問い合わせに対応するよう定義され，機能します。他アプリケーションと通信するには，アプリケーションはメッセージを定義済みキューに登録します。相手アプリケーションはキューからメッセージを受信し，メッセージに含まれている問い合わせと情報を処理します。つまり，メッセージキューイングはプログラム間通信です。

キューイングは，アプリケーションによる処理が可能になるまで，メッセージを保持する機能です。キューイングには，次に示す機能があります。

- 通信コードを記述する必要のないプログラム間通信
  各プログラムが異なる環境で動作できます。
- アプリケーションがメッセージを処理する順序の選択
- システム内の負荷分散
  メッセージ数がしきい値を超える場合に複数のアプリケーションで一つのキューに対応します。
- アプリケーションの可用性の向上
  主系システムが使用不能になると待機系システムがキューに対応します。

1.2　メッセージの概要

メッセージは相手プログラムに送信するデータの集合です。TP1/Message Queue では，次に示すメッセージタイプを定義します。

非問い合わせ
　応答を必要としない単純なメッセージ

問い合わせ
　応答を必要とするメッセージ

応答
　問い合わせメッセージに対する応答

報告
　エラーの発生のようなイベントを通知するメッセージ

各メッセージタイプについては，「3.3　メッセージタイプ」を参照してください。

1.2.1　制御情報としてのメッセージ記述子

メッセージは制御情報とアプリケーションデータで構成されます。制御情報は MQMD 構造体で定義されます。MQMD 構造体内には，次に示す情報などがあります。

- メッセージタイプ
- メッセージ識別子
- 転送時のメッセージ優先度

アプリケーションデータの構造と内容は，TP1/Message Queue ではなく，アプリケーションによって決められます。

1.2.2　メッセージチャネルエージェント

メッセージチャネルエージェントは，キューマネジャ間でメッセージを転送します。

1.3　メッセージキューの概要

メッセージキューは単にキューともいいます。メッセージが送信される名前付きのあて先です。キューに対応するアプリケーションによって取り出されるまで，メッセージはキューに蓄積されます。

キューマネジャのあるところにキューは配置され，キューマネジャによって管理されます。また，キューはキューファイル内に作成され，キューファイルの物理的なあり方はユーザによって定義されます。キューファイルの物理的な管理はキューマネジャが行い，関連するアプリケーションで意識する必要はありません。

キューマネジャのアプリケーションプログラミングインタフェースを介することによって，アプリケーションはキューにアクセスします。アプリケーションはキューをオープンしたり，メッセージをキューに登録したり，キューからメッセージを取り出したり，キューをクローズしたりできます。

1.4　キューマネジャの概要

キューマネジャは，アプリケーションにキューイングサービスを提供するシステムプログラムです。キューマネジャは，アプリケーションからメッセージをキューに登録したり取り出したりするためのアプリケーションプログラミングインタフェースを提供します。また，システム管理者が新規のキューを作成したり，既存のキューの属性を変更したり，キューマネジャを操作するための機能を提供します。

互いに無関係のアプリケーションが同時にキューマネジャの機能を利用できます。キューマネジャの機能を利用するアプリケーションは，キューマネジャへ接続しなくてはなりません。

1.5　クラスタの概要

クラスタは，論理的に関連づけられたキューマネジャのネットワークです。

クラスタを使わない，分散キューイングによるネットワークでは各キューマネジャは独立しています。相手システムにメッセージを送信するキューマネジャには，定義済みの転送キュー，相手キューマネジャへのチャネル，およびメッセージ送信先の各キューについてのリモートキューのローカル定義が必要です。

キューマネジャをグループ化してクラスタにする場合，各キューマネジャが保持するキューを，クラスタ内の各キューマネジャから利用できます。適切に構成されたクラスタのネットワークでは，キューマネジャからキューマネジャにメッセージを送信するのに，あて先ごとの転送キュー，チャネル，およびリモートキューのローカル定義は不要です。

クラスタを使用する長所は，システム管理の手間を軽減できることと，負荷分散の改善にあります。

小さなクラスタを設定する場合でもシステム管理は簡単になります。クラスタ内のキューマネジャでは定義が少なくなるので，ユーザが定義間違いをする可能性も減ります。

詳細については，マニュアル「TP1/Message Queue 使用の手引」を参照してください。

1.6　メッセージキューイングの特長

メッセージキューイングを使用するアプリケーションの特長について次に示します。

- アプリケーション間の直接的なコネクションがありません。
- アプリケーション間の通信が時間に依存しません。
- 小さなアプリケーションで業務を実行できます。
- 通信をイベントで起動できます。
- アプリケーションはメッセージに優先度を設定できます。
- 同期点をサポートします。
- メッセージの回復をサポートします。

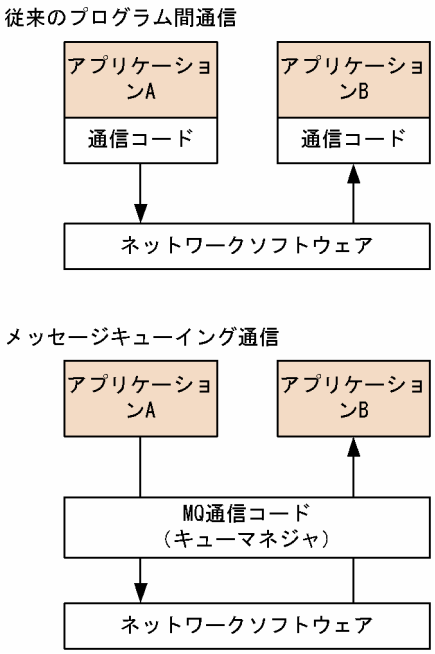
1.6.1　間接的なプログラム間通信

メッセージキューイングは間接的なプログラム間通信のための技術です。通信アプリケーション内のどこにでも使用できます。アプリケーションがメッセージをキューに登録し，相手アプリケーションがメッセージをキューから取り出すことによって，通信が発生します。

メッセージキューイングを使用して通信するプログラム間には，物理的なコネクションがありません。キューマネジャが保持するキューにアプリケーションからメッセージを送信し，相手アプリケーションはキューからメッセージを取り出します。

従来のプログラム間通信とメッセージキューイング通信との比較について，次の図に示します。

図 1-1　従来のプログラム間通信とメッセージキューイング通信との比較



各メッセージはトランザクションの一部であり，キューへの保存とチャネルによる転送によってネットワークを移動します。ノード間の接続に障害が発生しても，障害から回復するまでメッセージは保持されます。または，オペレータやプログラムによって，転送先を変更されます。

メッセージがキューからキューへと移動する仕組みについて，アプリケーションは意識する必要がありません。そのため，アプリケーションは単純化されます。

1.6.2 時間に依存しない通信

処理を要求するアプリケーションは，要求に対する応答を待ち合わせる必要がありません。ほかの作業を実行しながら，到着時以降に応答を処理できます。アプリケーションを作成するときには，いつメッセージを送信するのか，相手がいつメッセージを受信するのか意識しなくてかまいません。メッセージが失われることはなく，相手が処理できるようになるまでキューマネジャによって保持されます。メッセージが削除されるのは，アプリケーションによって削除されるときです。

1.6.3 小さなアプリケーション

メッセージキューイングによって，小さなアプリケーションの特長を利用できます。すべての業務を順に実行する単一の巨大なアプリケーションの代わりに，複数の独立した小さなアプリケーションに業務を分散できます。要求側のアプリケーションは各アプリケーションにメッセージを送信し，機能を実行するように依頼します。各アプリケーションが完了すると，結果がメッセージとして送り返されます。

1.6.4 イベント起動の処理

アプリケーションをキューの状態に従って制御できます。例えば，メッセージがキューに到着するのと同時に業務を開始するようにできます。または，特定の優先度以上の 10 メッセージがキューに到着してから業務を開始したり，優先度を無視して開始したりできます。

1.6.5 メッセージ優先度の設定

アプリケーションがメッセージをキューに登録するときに，メッセージに優先度を設定できます。メッセージ優先度によって，新規メッセージが追加されるときのキュー内部での位置が決定されます。

アプリケーションは，メッセージをキュー内部での出現順に取り出したり，特定メッセージを取り出したりできます。アプリケーションが特定メッセージを取り出すのは，以前に送信した問い合わせの応答を探すときなどです。

1.6.6　同期点のサポート

トランザクションへの参加が，各 MQGET 命令と MQPUT 命令のオプションでサポートされます。トランザクションへ参加すると，命令の結果はコミットやロールバックの対象となります。アプリケーションでの指定によって，同期点への参加と不参加を指定できます。

1.6.7　メッセージの回復

必要な情報がシステムジャーナルに取得されるので，永続メッセージを回復できます。永続メッセージについては，「3.7　メッセージ永続性」を参照してください。

1.7　アプリケーション設計者にとっての利点

メッセージキューイングを使用するアプリケーションを開発する場合の利点について次に示します。

- 複数のアプリケーションで共有できる小さなプログラムを使用するアプリケーションを作成できます。
- 作成済みの部品を再利用できるので新しいアプリケーションを簡単に作成できます。
- メッセージキューイング技術を使用して記述されたアプリケーションは，キューマネジャの動作変更による影響を受けません。
- アプリケーションは通信プロトコルを意識する必要がありません。キューマネジャに任せられます。
- メッセージが送信されるときに受信側のアプリケーションが動作している必要がありません。メッセージはキューに保持されます。

2 アプリケーション設計の概要

この章では，アプリケーション設計の概要について説明します。

2.1 設計計画

TP1/Message Queue が提供する機能をどのように利用するか決めてください。

次に示す検討項目があります。

使用するキュータイプ

必要になるたびにキューを作成するのか，設定済みのキューを使用するのか検討します。使用したあとにキューを削除するのか，再度使用するのか検討します。アプリケーションの独立性のために別名キューを使用するのか検討します。キュータイプについては，「4.3 キュー」を参照してください。

使用するメッセージタイプ

単純なメッセージには，非問い合わせメッセージを使用します。その他の応答が必要な状況では，問い合わせメッセージを使用します。特定メッセージに異なる優先度を設定することもできます。

データの一貫性の保持

OpenTP1 が提供するトランザクション機能を使用すると，その他のリソースとのデータの一貫性を保持できます。また，メッセージの重要性に応じて永続メッセージにするか非永続メッセージにするかを検討します。

例外とエラーへの対処

転送されなかったメッセージをどのように処理するか，また，キューマネジャによって報告されるエラー状況をどのように解決するか検討します。報告オプションには MQPUT 命令および MQPUT1 命令の Report オプションの設定が必要です。

以降では，ここで挙げた項目で TP1/Message Queue が提供する機能について説明します。

2.2　オブジェクトの使用

MQI は次に示す種類のオブジェクトを使用します。

- キューマネジャ
- キュー
- プロセス定義
- チャネル

各オブジェクトの詳細については，「4.　オブジェクト」を参照してください。

これらのオブジェクト（動的キューを除く）を使用する前にキューマネジャに定義しなければなりません。

オブジェクトを定義するには，次に示す定義方法があります。

- MQA サービス定義
- MQT サービス定義

これらの詳細については，マニュアル「TP1/Message Queue 使用の手引」を参照してください。

オブジェクトの属性は表示したり，変更したりできます。また，オブジェクトは削除できます。

2.3 メッセージの設計

MQI 命令を使用してメッセージをキューに登録するときに，ユーザはメッセージを作成します。命令への入力として，メッセージ記述子（MQMD 構造体）内の幾つかの制御情報，および相手アプリケーションに送信したいデータを設定します。しかし，設計時には，メッセージの作成方法に影響があることから，次に示す項目について検討してください。

使用するメッセージタイプ

単純なメッセージを送信するだけで以降の応答を期待しないのか，問い合わせに対する応答を期待するのかを検討します。問い合わせをする場合には，応答を受信するためのキューの名前をメッセージ記述子に設定してください。

また，問い合わせと応答のメッセージを同期させるか検討します。つまり，問い合わせに対する応答には期限を設定でき，それを過ぎるとエラーとして処理されます。

または，プロセスが共通タイミングシグナルのような特定イベントの発生に依存しないように，非同期に処理するのか検討します。

その他の検討項目としては，すべてのメッセージをトランザクション内で処理するかどうかということがあります。

作成するメッセージに異なる属性を設定するか

各メッセージには優先度の値を設定できます。また，キューが優先度に応じてメッセージを保持するよう定義できます。この場合には，相手アプリケーションがキューからメッセージを取り出すときには常に高い優先度のメッセージを取り出します。キューがメッセージ優先度を無視する場合，メッセージが登録された順序に応じて取り出されます。

メッセージがキューに登録されたときにキューマネジャによって付加されるメッセージ識別子を使用することによって，アプリケーションでメッセージを選択することもできます。また，ユーザ独自の識別子をメッセージに設定することもできます。

OpenTP1 の再開始時にメッセージを保持するか

キューマネジャはすべての永続メッセージを保持し，再開始するときに回復します。しかし，非永続メッセージと一時的動的キューは再開始するときには回復されません。失いたくないメッセージを作成するときには永続メッセージとして作成してください。

メッセージの受信側に自システムの情報を与えるべきか

ユーザが設定しない場合，キューマネジャがメッセージにユーザ識別子を設定します。受信側アプリケーションが課金やセキュリティの目的でユーザ識別子を使用する場合は，送信側アプリケーションでユーザ識別子を設定してください。

2.4　アプリケーションの手法

単純なアプリケーションでは，使用するオブジェクトとメッセージのタイプを決定しなければなりません。より複雑なアプリケーションでは，ここで説明するような手法を使用できます。

2.4.1　メッセージの待ち合わせ方法

キューに対応するアプリケーションは，次に示す方法によってメッセージを待ち合わせます。

- 定期的に MQI 命令を発行して，キューにメッセージが届いているかを確認（ポーリング）します。
- メッセージが到着するか，または指定した時間が経過するまで待ちます。詳細については，「10.6　メッセージの待ち合わせ」を参照してください。

2.4.2　応答の関連づけ

問い合わせメッセージを受信したアプリケーションは，通常，一つ以上の応答メッセージを問い合わせ元に送信することになります。問い合わせ元のアプリケーションが応答と元の問い合わせを関連づけやすくなるように，各メッセージのメッセージ記述子に相関識別子を設定できます。問い合わせメッセージのメッセージ識別子を，応答メッセージの CorrelId フィールドにコピーしてください。

2.4.3　コンテキスト情報の設定と利用

コンテキスト情報は，作者であるユーザまたはメッセージを生成したアプリケーションをメッセージと関連づけるために使用されます。この情報は，セキュリティ，課金，認証，および障害個所の決定に役立ちます。

メッセージを作成するときに，キューマネジャがデフォルトのコンテキスト情報をメッセージに関連づけるようにオプションで指定できます。

コンテキスト情報については，「3.12　メッセージコンテキスト」を参照してください。

2.4.4　報告の生成

次に示す報告をアプリケーションに要求できます。

- 例外報告
- 保持時間報告
- メッセージ到着確認（COA）報告

- メッセージ配布確認（COD）報告
- 肯定動作の通知（PAN）報告
- 否定動作の通知（NAN）報告

詳細については，「3.3.4　報告メッセージ」を参照してください。

2.4.5　クラスタとメッセージ類似性

同じキューの複数の定義を持ったクラスタを使用する前に，メッセージ類似性を持っていないか，つまり関連メッセージの交換要求がないか，自分のアプリケーションを調査しなければなりません。クラスタでは，該当するキューを保持しているキューマネジャにメッセージがルーティングされます。したがって，メッセージ類似性を持っているアプリケーションのロジックは無効になることがあります。

例えば，質疑応答の形での連続するメッセージフローに依存している二つのアプリケーションがあるとします。すべての質問を同じキューマネジャに送信し，すべての回答をほかのキューマネジャに送り返さなければいけないとします。この場合，ワークロード管理機能が，該当するキューを保持しているだけのキューマネジャにメッセージを送信しないようにしてください。

メッセージ類似性は取り除いてください。メッセージ類似性を取り除くとアプリケーションの可用性とスケーラビリティが改善されます。

詳細については，マニュアル「TP1/Message Queue 使用の手引」を参照してください。

2.5 アプリケーションプログラミング

TP1/Message Queue は IBM MQ の MQI 命令をサポートします。MQI 命令を使用すると，メッセージを送受信したりオブジェクトを操作したりできます。

2.5.1 命令インタフェース

MQI 命令では次に示す操作ができます。

- アプリケーションとキューマネジャの接続と切断
- オブジェクト（キュー，キューマネジャ，プロセスなど）のオープンとクローズ
- メッセージのキューへの登録
- メッセージのキューからの取り出し，およびキューに残したままの検索
- オブジェクト属性の照会および設定

MQI は命令に設定したり取り出したりできる構造体を提供します。また，命令のパラメタを設定するための名前付き定数を提供します。命令，構造体，および名前付き定数は各プログラミング言語が提供する定義ファイルで提供されます。また，デフォルトの値は MQI 命令内に設定されます。

2.5.2 性能向上のためのヒント

効率的なアプリケーションを設計するためのヒントについて説明します。

- ユーザがアプリケーションを動作させながら考えている時間と，並行して処理するようにアプリケーションを設計してください。

  異なるサーバから並行して必要なデータを取得してもかまいません。
- 再使用する場合は，オープンとクローズ，接続と切断を繰り返さないで，コネクションとキューをオープンしたままにしてください。

  ただし，1 メッセージだけを登録するサーバアプリケーションでは MQPUT1 命令を使用してください。
- 複数のメッセージ処理を一つのトランザクション内に記述し，同時にコミットまたはロールバックするようにしてください。
- 回復が不要なメッセージには非永続メッセージのオプションを使用してください。

2.5.3 複数 OS 向けのアプリケーション

アプリケーションを複数の OS で動作させる場合には，プログラムコードが OS に依存しないように作成してください。

C 言語を使用する場合，OS に固有の関数が高速であっても，標準ライブラリを使用してください。処理速度が重要な場合は，#ifdef を使用して両方のコードを記述してください。

例えば，次に示すとおり記述します。

```
#ifdef _HPUX
    HP-UX固有のコード
#else
    標準のコード
#endif
```

プログラムコードをほかの OS に移植する場合，#ifdef を検索して必要なコードを追加したり変更したりできます。

プログラムコードを OS に依存しないようにすることと，単純な命名規則を使用することを心掛けてください。

3 メッセージ

この章では，メッセージについて説明します。

3.1　メッセージの構成

TP1/Message Queue のメッセージは次に示す要素から構成されます。

- メッセージ記述子
- アプリケーションデータ

メッセージの構成について，次の図に示します。

図 3-1　メッセージの構成

メッセージによって転送されるアプリケーションデータは，データ変換を除くと，キューマネジャによって変更されることはありません。また，TP1/Message Queue によって内容が制限されることもありません。各メッセージのデータ長は，キューおよびキューマネジャの MaxMsgLength 属性値を超えられません。TP1/Message Queue では MaxMsgLength 属性のデフォルト値は 4096000 バイトです。詳細については，「9.2.4　ユーザメッセージ内のデータ」を参照してください。

ユーザは MQPUT 命令または MQPUT1 命令を使用するときにメッセージを作成します。これらの命令への入力には，制御情報（メッセージ優先度や応答キュー名など）とユーザデータを設定します。これらの命令はメッセージをキューに登録する命令です。詳細については，マニュアル「TP1/Message Queue プログラム作成リファレンス」を参照してください。

3.2　メッセージ記述子

メッセージ記述子を定義する MQMD 構造体を使用して，メッセージの制御情報にアクセスできます。MQMD 構造体の詳細については，マニュアル「TP1/Message Queue プログラム作成リファレンス」を参照してください。

メッセージの登録元についての情報を含む MQMD 構造体内のフィールドの使用方法については，「3.12 メッセージコンテキスト」を参照してください。

複数のバージョンのメッセージ記述子が定義されています。メッセージのグループ化とセグメント分割についてのフィールドがバージョン 2 の構造体でサポートされました。バージョン 2 の構造体は，バージョン 1 の構造体とほぼ同じですが，MQMDE 構造体にあるフィールドがあります。詳細については，「3.6 メッセージグループ」を参照してください。

3.3 メッセージタイプ

TP1/Message Queue では，次に示すメッセージタイプを使用できます。

- 非問い合わせ
- 問い合わせ
- 応答
- 報告

アプリケーション間で情報を受け渡しするには，非問い合わせ，問い合わせ，および応答を使用します。報告は，エラー発生のようなイベントについての報告情報を使用するアプリケーションおよびキューマネジャで使用します。

各メッセージタイプは MQMT_*の値で識別します。ユーザが独自のメッセージタイプを定義することもできます。設定できる値の範囲については，MQMD 構造体の MsgType フィールドの説明を参照してください。

3.3.1 非問い合わせメッセージ

メッセージを受信するアプリケーションからの応答を必要としない場合は，非問い合わせメッセージを使用します。

非問い合わせメッセージを使用するアプリケーションの例としては，空港のラウンジでフライト情報を表示するアプリケーションがあります。非問い合わせメッセージはフライト情報の 1 画面分のデータです。このようなアプリケーションでは，メッセージが転送されなかったとしても問題にならないので，メッセージに対する肯定応答を要求しません。すぐあとに最新メッセージを送信できるからです。

3.3.2 問い合わせメッセージ

メッセージを受信するアプリケーションからの応答を必要とする場合は，問い合わせメッセージを使用します。

問い合わせメッセージを使用するアプリケーションの例としては，預金残高を表示するアプリケーションがあります。問い合わせメッセージには口座番号のデータがあります。応答メッセージには預金残高のデータがあります。

応答メッセージを問い合わせメッセージに関連づけるには，次に示す方法があります。

- 問い合わせメッセージを処理するアプリケーションで，関連する応答メッセージに情報を登録する方法
- 問い合わせメッセージのメッセージ記述子にある Report フィールドを使用して，応答メッセージの MsgId および CorrelId フィールドの内容を指定する方法

- 問い合わせメッセージの MsgId または CorrelId フィールドを，応答メッセージの CorrelId フィールドにコピーするように指定できます。デフォルトでは MsgId フィールドをコピーします。
- 新規の MsgId フィールドを応答メッセージのために生成するか，または問い合わせメッセージの MsgId フィールドを応答メッセージの MsgId フィールドにコピーするかを指定できます。デフォルトでは新規の MsgId フィールドを生成します。

3.3.3 応答メッセージ

ほかのメッセージに応答する場合は，応答メッセージを使用します。

応答メッセージを作成するときは，受信したメッセージのメッセージ記述子に設定されたオプションに注意してください。報告オプションはメッセージ識別子（MsgId フィールド）および相関識別子（CorrelId フィールド）の内容を指定します。これらのフィールドによって，応答を受信するアプリケーションは応答と元の問い合わせを関連づけます。

3.3.4 報告メッセージ

報告メッセージは，メッセージ処理時のエラー発生などのイベントを，アプリケーションに知らせます。報告メッセージは次に示すプロセスによって生成されます。

- キューマネジャ
- メッセージチャネルエージェント（メッセージを転送できないときなど）
- アプリケーション（メッセージ内のデータを使用できないときなど）

報告メッセージが生成されるタイミングは任意であり，キューに登録されます。

(1) 報告メッセージの種類

メッセージをキューに登録するとき，次に示すメッセージの受信を選択できます。

- 例外報告メッセージ。このメッセージは例外フラグが設定されたメッセージに対して送信されます。メッセージチャネルエージェントまたはアプリケーションによって生成されます。
- 保持時間報告メッセージ。保持時間が経過したメッセージをアプリケーションで取り出そうとしたことを示します。メッセージは破棄されます。この報告はキューマネジャによって生成されます。
- メッセージ到着確認（COA）報告メッセージ。メッセージがあて先のキューに到着したことを示します。キューマネジャによって生成されます。
- メッセージ配布確認（COD）報告メッセージ。受信側アプリケーションによってメッセージが取り出されたことを示します。キューマネジャによって生成されます。
- 肯定動作の通知（PAN）報告メッセージ。メッセージで要求した動作が実行されたことを示します。この報告はアプリケーションによって生成されます。

- 否定動作の通知（NAN）報告メッセージ。メッセージで要求した動作が実行されなかったことを示します。この報告はアプリケーションによって生成されます。

注意

各種の報告メッセージは，次に示すどれかに該当します。

- 元のメッセージを持っています。
- 元のメッセージのデータから先頭 100 バイトを持っています。
- 元のメッセージからのデータを持っていません。

メッセージをキューに登録するときには，1 種類以上の報告メッセージを要求できます。メッセージ配布確認（COD）報告メッセージと例外報告メッセージのオプションを選択した場合，メッセージの転送が失敗すると例外報告メッセージを受信できます。しかし，メッセージ配布確認（COD）報告メッセージのオプションだけを選択した場合には，メッセージの転送が失敗しても例外報告メッセージを受信できません。

該当するメッセージを生成するための選択基準が満たされるとき，要求した報告メッセージだけを受信できます。

(2) 報告メッセージオプション

報告メッセージを受信する場合には，ReplyToQ フィールドに応答キューの名前を指定してください。指定しない場合，元のメッセージを発行する MQPUT 命令と MQPUT1 命令は MQRC_MISSING_REPLY_TO_Q で失敗します。

該当するメッセージに対して生成される報告メッセージの MsgId および CorrelId フィールドの内容を指定するために，メッセージ記述子（MQMD 構造体）にその他の報告オプションを設定することもできます。

- 元のメッセージの MsgId または CorrelId フィールドを，報告メッセージの CorrelId フィールドにコピーするように指定できます。デフォルトでは MsgId フィールドをコピーします。MQRO_COPY_MSG_ID_TO_CORRELID を指定することをお勧めします。これによって，メッセージの送信側が，応答メッセージまたは報告メッセージを元のメッセージと関連づけることができます。応答メッセージまたは報告メッセージの CorrelId フィールドは，元のメッセージの MsgId フィールドと同じになります。
- 新規の MsgId フィールドを報告メッセージのために生成するか，または元のメッセージの MsgId フィールドを報告メッセージの MsgId フィールドにコピーするか指定できます。デフォルトでは新規の MsgId フィールドを生成します。MQRO_NEW_MSG_ID を指定することをお勧めします。これによって，システム内の各メッセージが異なる MsgId フィールドを持つことになり，システム内の他メッセージと明確に区別できます。
- 特殊なアプリケーションでは MQRO_PASS_MSG_ID，MQRO_PASS_CORREL_ID または両方を指定する必要があります。しかし，キューからメッセージを取り出すアプリケーションでは，正常に処理するために配慮が必要です。キューに同一の MsgId フィールドで複数のメッセージが格納されている場合は特に必要です。

サーバアプリケーションは，問い合わせメッセージにあるこれらのフラグの設定を確認し，応答メッセージまたは応答メッセージの MsgId および CorrelId フィールドを適切に設定する必要があります。

問い合わせアプリケーションとサーバアプリケーションを中継するアプリケーションは，これらのフラグの設定を確認する必要はありません。通常は，サーバアプリケーションに転送する際に，MsgId，CorrelId および Report フィールドを変更しないようにする必要があるからです。これによって，サーバアプリケーションは，元のメッセージの MsgId フィールドを応答メッセージの CorrelId フィールドにコピーできます。

メッセージについての報告を作成するときには，サーバアプリケーションでこれらのオプションが設定されているか確認するようにしてください。

報告の性質を示すために，キューマネジャは報告メッセージ返答コードを使用します。キューマネジャは報告メッセージ返答コードを報告メッセージのメッセージ記述子の Feedback フィールドに設定します。また，MQI の理由コードを Feedback フィールドに返すこともあります。TP1/Message Queue ではアプリケーションで使用できる報告メッセージ返答コードを規定しています。

報告メッセージ返答コードを使用するアプリケーションの例としては，他アプリケーションがキューを処理する負荷を監視するアプリケーションがあります。一つのキューを処理している複数のアプリケーションがあるとします。キューにあるメッセージの数がそれだけの数のアプリケーションを必要としなくなると，報告メッセージ（Feedback フィールドが MQFB_QUIT）をアプリケーションに送信して，動作を中止するよう指示できます。幾つのアプリケーションがキューを処理しているのか検出するために，監視アプリケーションは MQINQ 命令を使用できます。

3.3.5　報告メッセージとセグメント分割メッセージ

セグメント分割メッセージは TP1/Message Queue 05-00 以降でサポートします。セグメント分割メッセージについては，「10.5.2　メッセージのセグメント分割」を参照してください。

メッセージがセグメント分割されているときに報告メッセージを生成すると，セグメント分割されていないときよりも多くの報告メッセージを受け取ることになります。

(1)　システムで生成される報告メッセージ

メッセージをセグメント分割する場合，またはキューマネジャに分割させる場合で，完全なメッセージについて単一の報告メッセージを受け取る状況は一つだけです。

それは，メッセージ配布確認（COD）報告だけを要求し，メッセージを取り出すアプリケーションで MQGMO_COMPLETE_MSG を指定している状況です。

その他の状況では，セグメントごとに報告メッセージが発生するので，アプリケーションで複数の報告メッセージに対応できるよう準備してください。

注意

メッセージをセグメント分割するときに元のメッセージの先頭 100 バイトだけが返されるようにしたい場合には，100 バイト目以上のオフセットに当たるセグメントでデータなしの報告メッセージを要求するよう，報告オプションの設定を変更してください。各セグメントが 100 バイトのデータを要求する設定のまま MQGMO_COMPLETE_MSG オプションを指定した MQGET 命令を発行して報告メッセージを取り出そうとすると，各オフセットで 100 バイトの読み込みデータが含まれる長大な報告メッセージが生成されます。その場合には，長大なバッファを用意するか，または MQGMO_ACCEPT_TRUNCATED_MSG オプションを指定してください。

(2) アプリケーションで生成される報告メッセージ

アプリケーションで報告メッセージを生成する場合は，元のメッセージデータの先頭にあるヘッダを報告メッセージデータにコピーしてください。その後ろには，元のメッセージデータ（またはその他のユーザデータ）のすべてを追加したり，100 バイトを追加したり，何も追加しないようにします。

MQMD 構造体の Format フィールドを参照して，どのヘッダをコピーすればよいか判断してください。連続する複数のヘッダが設定されていることがあります。

次に示すフォーマット名はヘッダを表します。

- MQMDE
- MQDLH
- MQXQH
- MQH*（"MQH"で始まる名前のことです）

Format フィールドは MQDLH 構造体および MQXQH 構造体では独自の位置にあります。しかし，その他のヘッダは同じ位置にあります。

セグメント，グループ内のメッセージ，またはセグメント分割を許可されたメッセージで，バージョン 1 の MQMD 構造体を使用して報告する場合，報告データは MQMDE 構造体で始まるようにしてください。OriginalLength フィールドには，元のメッセージデータの長さを設定してください。この長さには検出したヘッダの長さを含みません。

(3) 報告メッセージの取り出し

メッセージ到着確認（COA）報告メッセージ，またはメッセージ配布確認（COD）報告メッセージを要求した場合，MQGMO_COMPLETE_MSG オプションを指定するとメッセージが組み立てられます。MQGMO_COMPLETE_MSG オプションを指定した MQGET 命令は，一つの完全な元のメッセージを表すのに十分な報告メッセージ（単一のタイプで同じ GroupId）がキューにあると，成功します。報告メッセージ自身に完全な元のデータを含まない場合でも成功します。その場合は，データそのものがなくても各報告メッセージの OriginalLength フィールドから，元のデータの長さがわかります。

この方法は幾つかの異なるタイプの報告メッセージがキューにある場合にも使用できます。MQGMO_COMPLETE_MSG オプションを指定した MQGET 命令では，複数の報告メッセージで同じ

Feedback コードを持てば報告メッセージを組み立てることができるからです。しかし，例外報告メッセージでは異なる Feedback コードを持つことが多いので，通常はこの方法を使用できないことに注意してください。

この方法は完全なメッセージが到着したという肯定指示を取得するのに使用できます。しかし，到着したセグメントがある一方で幾つかのセグメントで例外（または保持時間の超過）が発生している可能性を考えなければいけないことがあります。上記のとおり，異なる Feedback コードがセグメントごとに設定されていることが多いので，通常は MQGMO_COMPLETE_MSG オプションを指定した MQGET 命令は使用できません。該当するセグメントについて，複数の報告メッセージを取得することになります。しかし，これについては MQGMO_ALL_SEGMENTS_AVAILABLE オプションを使用する方法があります。

以上のことから，到着時に報告メッセージを取り出して，元のメッセージに何が発生しているのかアプリケーションで判断するようにしてください。報告メッセージの GroupId フィールドを使用して元のメッセージと報告メッセージを関連づけることができます。Feedback フィールドを使用して報告メッセージの種類を識別できます。どの方法を使うかはアプリケーションの要件によって異なります。

対処方法の例を次に示します。

- メッセージ配布確認（COD）報告メッセージ，および例外報告メッセージを要求します。
- 一定の時間のあとに，MQGMO_COMPLETE_MSG オプションを使用して完全なメッセージ配布確認（COD）報告メッセージが受信されたかを確認します。受信された場合には，メッセージ全体が処理されたことをアプリケーションで確認できます。
- 配布確認（COD）報告メッセージが受信されないで，該当するメッセージに関連する例外報告メッセージがある場合は，セグメント分割されないメッセージと同様に処理します。孤立したセグメントを消去するという準備も同時に必要です。
- どの種類の報告メッセージもないセグメントがある場合は，元のセグメント（または報告メッセージ）がチャネルの再接続を待っているか，またはネットワークがその時点で過負荷になっていることが考えられます。例外報告メッセージがまったく受信されていない場合には（または一時的なものだけを受信していると考えられる場合），アプリケーションをしばらく待機させてください。

  上記と同様に，これはセグメント分割されないメッセージを処理するときの検討項目と同じです。しかし，孤立したセグメントを消去するという準備も同時に必要です。

あとから再送が可能な問い合わせメッセージのように，元のメッセージが重要でない場合には，孤立したセグメントが消去されるようにメッセージ保持時間を設定してください。

(4)　セグメント分割をサポートしないキューマネジャ

セグメント分割をサポートするキューマネジャで報告メッセージを生成し，サポートしないキューマネジャで受信するときには，MQMDE 構造体（報告メッセージの Offset および OriginalLength フィールドを識別する）が，0 バイト，100 バイト，または完全なメッセージの元のデータとともに，報告データに含まれます。

しかし，セグメント分割をサポートしないキューマネジャを，メッセージのセグメントが経由する場合には，元のメッセージにある MQMDE 構造体がデータとして扱われるのか注意が必要です。元のデータの 0 バイトが要求される場合には，報告データには MQMDE 構造体は含まれません。MQMDE 構造体がないと報告メッセージは使用できません。

そこで，セグメント分割をサポートしないキューマネジャをメッセージが経由する可能性のあるときには，報告メッセージに少なくともデータの 100 バイトを含むよう要求してください。

3.4　メッセージ制御情報とメッセージデータの形式

メッセージを処理するアプリケーションは，制御情報とデータの両方を認識しますが，キューマネジャが認識するのはメッセージ内にある制御情報の形式だけです。

3.4.1　制御情報の形式

メッセージ記述子の文字列フィールドにある制御情報は，キューマネジャが使用する文字セットにしてください。キューマネジャオブジェクトの CodedCharSetId 属性がこの文字セットを定義します。制御情報はこの文字セットで記述してください。メッセージをキューマネジャ間で受け渡しするときに，転送を担当するメッセージチャネルエージェントがデータ変換の方法を決定するためにこの属性を参照するからです。

3.4.2　メッセージデータの形式

次に示すデータを指定できます。

- アプリケーションデータの形式（Format フィールド）
- 文字データの文字セット（CodedCharSetId フィールド）
- 数値データの形式（Encoding フィールド）

各フィールドについて説明します。

Format

このフィールドは，メッセージの受信側にメッセージ内のアプリケーションデータの形式を示します。

キューマネジャがメッセージを作成するときに，Format フィールドを使用してメッセージの形式を識別することがあります。例えば，キューマネジャがメッセージを転送できないときに，メッセージをデッドレターキューに登録しますが，そのことを示すためにキューマネジャはメッセージにヘッダ（制御情報を含む）を追加して，Format フィールドを変更します。

キューマネジャは MQFMT_STRING のように"MQ"で始まる組み込みフォーマット名を持っています。それで不十分な場合は，ユーザは独自のフォーマットを設定できます。ただし，"MQ"で始まる名称は使用しないでください。

ユーザ独自の文字セットを作成して使用するときには，MQGMO_CONVERT オプションでメッセージを取り出すアプリケーションのために，データ変換をする UOC を記述してください。

CodedCharSetId

このフィールドは，メッセージ内にある文字データの文字セットを定義します。キューマネジャの文字セットを設定したい場合には，このフィールドは MQCCSI_Q_MGR にしてください。

メッセージをキューから取り出すときには，CodedCharSetId フィールドの値を，アプリケーションで処理できる値と比較してください。二つの値が異なる場合には，メッセージ内の文字データを変換したり，データ変換をする UOC を使用したりしてください。

Encoding

このフィールドは，2 進整数，10 進パック形式整数，および浮動小数点の数値メッセージデータのマシンコード形式を定義します。これらのデータは通常，キューマネジャが動作する特定のコンピュータの形式にエンコードされます。

メッセージをキューに登録するとき，通常は MQENC_NATIVE を Encoding フィールドに指定します。これはアプリケーションが動作しているコンピュータの形式にメッセージデータをエンコードするということです。

メッセージをキューから取り出すときには，メッセージ記述子にある Encoding フィールドの値を自コンピュータの MQENC_NATIVE の値と比較してください。二つの値が異なる場合には，メッセージ内の数値データを変換したり，データ変換をする UOC を使用したりしてください。

3.4.3 アプリケーションデータの変換

異なる OS と通信するときには，アプリケーションデータを相手アプリケーションが必要とする文字セットに変換してください。送信側のキューマネジャで変換したり，受信側のキューマネジャで変換したりできます。組み込みフォーマットのライブラリで不十分な場合には，独自のライブラリを作成してください。MQMD 構造体の Format フィールドで指定されたメッセージ形式に変換されます。

注意事項

- MQFMT_NONE が指定されたメッセージは変換されません。
- PCF（プログラマブルコマンドフォーマット）データの転送時は，コード変換（mqtalccha 定義コマンド-d オプションの cnvccsid オペランドに変換後の文字セット識別子を指定）は有効になりません。

(1) 送信側キューマネジャでの変換

送信側メッセージチャネルエージェントでアプリケーションデータを変換する場合には，mqtalccha 定義コマンド-d オプションの cnvccsid オペランドに変換後の文字セット識別子を指定します。

また，適切な UOC を作成すると送信側キューマネジャで組み込みフォーマットにデータ変換できます。

組み込みフォーマット

次に示すデータのフォーマットです。

- 文字（MQFMT_STRING のフォーマット名を使用）から構成されるメッセージ
- プログラマブルコマンドフォーマットのようなキューマネジャで定義するメッセージ

TP1/Message Queue ではメッセージとイベントを管理するためにプログラマブルコマンドフォーマットメッセージを使用します。この場合，フォーマット名は MQFMT_ADMIN です。ユーザは独自のメッセージに同じフォーマットを使用して，組み込みフォーマットを利用できます。この場合，フォーマット名は MQFMT_PCF です。

すべてのキューマネジャ組み込みフォーマットは"MQFMT"で始まる名前を持ちます。MQMD 構造体の Format フィールドについては，マニュアル「TP1/Message Queue プログラム作成リファレンス」を参照してください。

アプリケーション定義フォーマット

ユーザ定義フォーマットの場合，メッセージ編集出口 UOC でデータ変換できます。メッセージ編集出口 UOC については，マニュアル「TP1/Message Queue 使用の手引」を参照してください。

(2) 受信側キューマネジャでの変換

アプリケーションメッセージデータを，受信側キューマネジャで組み込みフォーマットに変換できます。MQGMO_CONVERT オプションが MQGET 命令に指定されると変換が実行されます。MQGMO 構造体については，マニュアル「TP1/Message Queue プログラム作成リファレンス」を参照してください。

文字セット識別子

TP1/Message Queue は動作環境の OS で提供される文字セット識別子をサポートします。キューマネジャを作成するときは，動作環境の OS の文字セット識別子が使用されます。

OS がサポートする文字セットについては OS のマニュアルを参照してください。

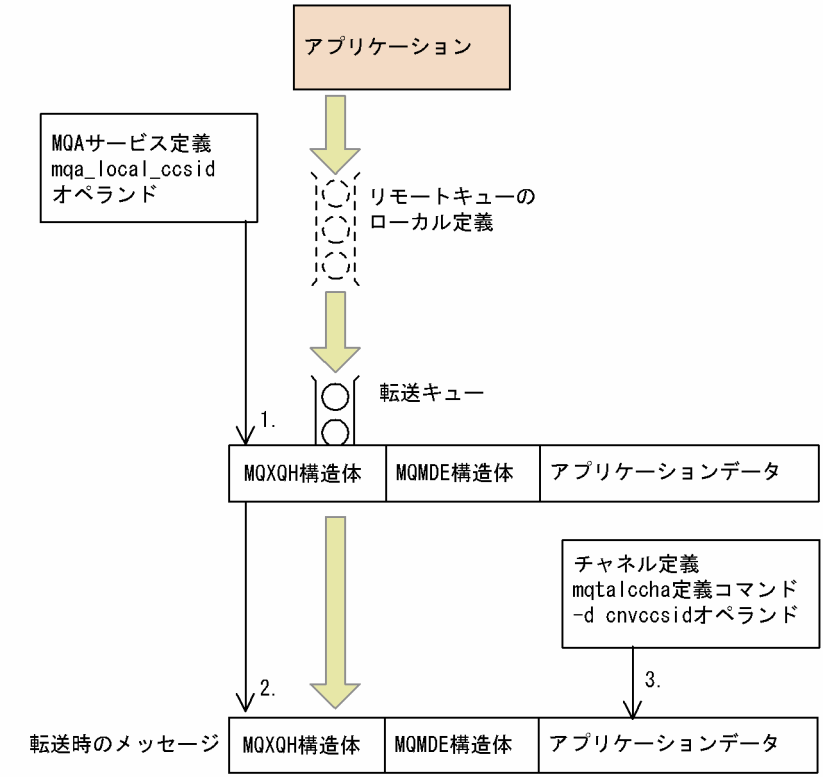
複数の OS にわたるアプリケーションを記述するときには，アプリケーションでのデータ変換，フォーマット名，および UOC について検討してください。

3.4.4 メッセージ形式の設定例

TP1/Message Queue が使用するメッセージの文字セット識別子およびマシンコード形式は，先頭に付加される MQ 構造体によって示されます。ここでは，アプリケーションがメッセージを転送キューに登録し，相手システムに送信されるまでのメッセージ形式の設定例について説明します。

メッセージ形式の設定例について，次の図に示します。

図 3-2　メッセージ形式の設定例



図中の番号について説明します。

1. アプリケーションが MQPUT 命令または MQPUT1 命令でメッセージを登録します。

   このとき，MQMD 構造体の CodedCharSetId および Encoding フィールドの指定を省略したとします。転送キュー上のメッセージにある MQXQH 構造体は，次に示す形式で作成されます。

   - 文字セット識別子

     キューマネジャの文字セット識別子（MQA サービス定義の mqa_local_ccsid オペランド指定値）
   - マシンコード形式

     キューマネジャが動作するコンピュータのマシンコード形式

   MQXQH 構造体に格納されている MQMD 構造体のフィールドは，次に示す値が設定されます。

   - CodedCharSetId フィールド

     MQCCSI_Q_MGR（0L，X'00000000'）が設定されます。
   - Encoding フィールド

     MQENC_NATIVE（計算機固有のマシンコード形式を表す値）が設定されます。

   アプリケーションが MQMD 構造体の Version フィールドに MQMD_VERSION_2 を指定した場合には，MQMDE 構造体が MQXQH 構造体の後ろに付加されます。付加される条件については，マニュアル「TP1/Message Queue プログラム作成リファレンス」を参照してください。

   MQXQH 構造体に付加される MQMDE 構造体のフィールドは，次に示す値が設定されます。

   - CodedCharSetId フィールド

MQCCSI_Q_MGR（0L，X'00000000'）が設定されます。

- Encoding フィールド
  MQENC_NATIVE（計算機固有のマシンコード形式を表す値）が設定されます。

2. MQ 構造体が変換されます。
   メッセージは転送キューから取り出され相手システムに送信されます。チャネルの開始時に自システム側でコード・ページ変換が必要になった場合，MQ 構造体（図 3-2 では MQXQH および MQMDE 構造体）はコード・ページ変換判定結果の文字セット識別子およびマシンコード形式で作成されます。
3. アプリケーションデータが変換されます。
   文字コード変換後の文字セット識別子（mqtalccha 定義コマンドの-d cnvccsid オペランド指定値）が送信側チャネルに指定されている場合，アプリケーションデータは該当する文字セットに変換されます。このとき，MQXQH 構造体に格納されている MQMD 構造体の CodedCharSetId フィールドは変換後の文字セット識別子に変更されます。
   MQMDE 構造体があるときは，MQMD 構造体ではなく MQMDE 構造体の CodedCharSetId フィールドが，変換後の文字セット識別子に変更されます。

図中の番号とメッセージの変更内容について，次の表に示します。

表 3-1　メッセージの変更内容

図中の番号	条件		メッセージの変更内容						
			MQXQH 構造体の作成時に使用する形式		MQXQH 構造体内の MQMD 構造体のフィールド		MQMDE 構造体のフィールド		アプリケーションデータ
	文字コード変換指定※1	MQMDE 構造体の有無	文字セット識別子	マシンコード形式	CodedCharSetId	Encoding	CodedCharSetId	Encoding	文字セット識別子
1.	−	なし	キューマネジャの文字セット識別子	コンピュータのマシンコード形式	MQCCSI_Q_MGR	MQENC_NATIVE	−	−	登録時の文字セット識別子
1.	−	あり	キューマネジャの文字セット識別子	コンピュータのマシンコード形式	MQCCSI_Q_MGR	MQENC_NATIVE	MQCCSI_Q_MGR	MQENC_NATIVE	登録時の文字セット識別子

図中の番号	条件		メッセージの変更内容						
			MQXQH 構造体の作成時に使用する形式		MQXQH 構造体内の MQMD 構造体のフィールド		MQMDE 構造体のフィールド		アプリケーションデータ
	文字コード変換指定※1	MQMDE 構造体の有無	文字セット識別子	マシンコード形式	Coded CharSetId	Encoding	Coded CharSetId	Encoding	文字セット識別子
2.	－	－	判定結果※2	判定結果※2	判定結果※2	判定結果※2	△	△	△
3.	なし	なし	△	△	△	△	－	－	△
3.	なし	あり	△	△	△	△	△	△	△
3.	あり	なし	△	△	変換後の文字セット識別子	△	－	－	変換後の文字セット識別子
3.	あり	あり	△	△	△	△	変換後の文字セット識別子	△	変換後の文字セット識別子

（凡例）

－：該当しません。

△：以前の状態を引き継ぎます。

注※1

mqtalccha 定義コマンドの-d cnvccsid オペランドの指定です。

注※2

チャネルの開始時に自システム側でコード・ページ変換が必要になった場合，コード・ページ変換の判定結果で作成されます。

3.5 メッセージ優先度

メッセージをキューに登録するときにメッセージの優先度（MQMD 構造体の Priority フィールド）を指定します。メッセージ優先度の数値を指定するか，またはキューのデフォルトの優先度を指定します。

キューの MsgDeliverySequence 属性には，メッセージが FIFO でキューに格納されるか，優先度付きの FIFO で格納されるかを設定します。MsgDeliverySequence 属性が MQMDS_PRIORITY に設定されると，メッセージはメッセージ記述子の Priority フィールドに指定された優先度でキューに格納され，MQMDS_FIFO に設定されるとキューのデフォルトの優先度で格納されます。同じ優先度のメッセージは到着順にキューに格納されます。

キューの DefPriority 属性には，キューに登録されるメッセージのデフォルトの優先度を設定します。キューを作成するときにこの属性を設定しますが，あとから変更できます。別名キュー，およびリモートキューのローカル定義は，基となるベースキューとは異なる優先度を設定できます。

キューマネジャの MaxPriority 属性には，キューマネジャによって処理されたメッセージに設定される優先度の最大値を設定します。ユーザはこの属性の値を変更できません。TP1/Message Queue ではこの属性の値は 9 です。ユーザは 0（最低）から 9（最高）までの優先度を持つメッセージを作成できます。

3.6 メッセージグループ

メッセージグループは TP1/Message Queue 05-00 以降でサポートします。

メッセージはグループ内にあります。これによって，メッセージの順序付けができます。また，同じグループ内では長大メッセージのセグメント分割ができます。

グループ内での階層を次に示します。

グループ

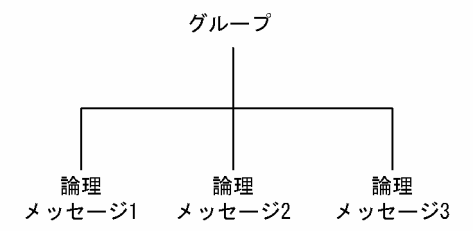
GroupId によって識別される最高階層です。同じ GroupId を持つ一つ以上のメッセージから構成されます。これらのメッセージはキューの任意の場所に登録されます。

注意

メッセージという用語は，ここではキュー上の 1 項目を表します。例えば，MQGMO_COMPLETE_MSG を指定しない MQGET 命令 1 回で返される項目です。

論理メッセージのグループについて，次の図に示します。

図 3-3 論理メッセージのグループ



論理メッセージ

グループ内の論理メッセージは GroupId および MsgSeqNumber フィールドによって識別されます。MsgSeqNumber はグループ内の最初のメッセージ 1 から始まります。グループ内にメッセージがない場合にもフィールドの値は 1 になります。

グループ内の論理メッセージは次に示す目的に使用できます。

- 順序性の保証（メッセージが転送される環境で保証されないとき）
- アプリケーションによる同種メッセージ（例えば，同じサーバインスタンスで処理したいメッセージ）のグループ化

セグメント分割されていないかぎり，グループ内の各メッセージは一つの物理メッセージで構成されます。各メッセージは論理的には別メッセージであり，MQMD 構造体の GroupId および MsgSeqNumber フィールドだけでグループ内の他メッセージとの関係を持つことになります。MQMD 構造体の他フィールドは関係ありません。すべてのメッセージで同じ値のフィールドもあるし，異なる値のフィールドもあります。例えば，一つのグループのメッセージで異なるフォーマット名，CCSID，マシンコード形式などを持つことがあります。

セグメント

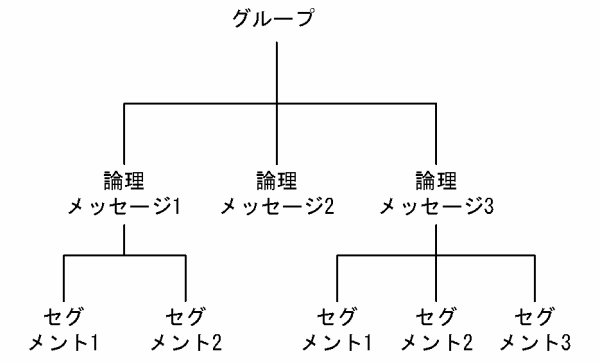
メッセージを登録したり取り出したりするアプリケーションやキューマネジャ（メッセージの転送経路のキューマネジャを含む）にとって，長過ぎるメッセージを処理するのに，セグメントが使用されます。詳細については，「10.5.2　メッセージのセグメント分割」を参照してください。

メッセージのセグメントは GroupId，MsgSeqNumber，および Offset フィールドで識別されます。Offset フィールドはメッセージ内の最初のセグメント 0 から始まります。

各セグメントは一つの物理メッセージで構成され，グループに属したり属さなかったりします。セグメントは論理的には 1 メッセージの部分です。そのため，同じメッセージの異なるセグメントでは，MQMD 構造体の MsgId，Offset，および MsgFlags フィールドだけが異なります。

セグメント分割された論理メッセージのグループについて，次の図に示します。

図 3-4　セグメント分割された論理メッセージのグループ



論理メッセージおよび物理メッセージについては，「10.3.2　論理的順序と物理的順序」を参照してください。メッセージのセグメント分割については，「10.5.2　メッセージのセグメント分割」を参照してください。

3.7　メッセージ永続性

永続メッセージは，システムジャーナルやキューファイルに書き出されます。システムジャーナルへの書き出しは TP1/Message Queue の環境に依存します。キューマネジャを障害後に再開始する場合，永続メッセージはシステムジャーナルから必要に応じて回復されます。キューマネジャが終了すると，オペレータコマンドの結果であるのか，システムの障害であるのかに関係なく，永続でないメッセージは破棄されます。

メッセージを作成するときにメッセージ記述子をデフォルトで初期化すると，MQOPEN 命令で指定されたキューの DefPersistence 属性にあるメッセージ永続性が使用されます。また，MQMD 構造体の Persistence フィールドを使用して，ユーザがメッセージの永続性を指定することもできます。

永続メッセージを使用する場合にはアプリケーションの性能に影響があります。影響範囲は，使用するコンピュータの I/O サブシステムと各 OS での同期点の使用方法によって異なります。

- TP1/Message Queue の仕掛かり中のトランザクション外部にある永続メッセージは，登録または取り出し操作のたびにディスクに書き出されます。詳細については，「12.　コミットとロールバック」を参照してください。
- TP1/Message Queue の仕掛かり中のトランザクション内部にある永続メッセージは，トランザクションがコミットした場合にだけシステムジャーナルおよびキューファイルに書き出されます。システムジャーナルへの書き出しは TP1/Message Queue の環境に依存します。トランザクションには複数のキュー操作を入れることができます。トランザクションに同期しない永続メッセージは，MQI 命令が正常終了した場合，システムジャーナルおよびキューファイルに書き出されます。

非永続メッセージは同期点外で取り出しを実行するときのファーストメッセージ機能で使用できます。ファーストメッセージ機能については，マニュアル「TP1/Message Queue 使用の手引」を参照してください。

3.8　キューからのメッセージの選択

キューから特定メッセージを取り出すには，メッセージ記述子の MsgId および CorrelId フィールドを使用します。バージョン 2 の MQMD 構造体を指定する場合には，GroupId も使用できます。

メッセージ識別子は，メッセージがキューに登録されるときにキューマネジャによって生成されます。キューマネジャはメッセージ記述子がユニークであるように保証します。しかし，アプリケーションでメッセージ記述子に特定の値を指定することができます。

ユーザは任意の相関識別子を使用できます。しかし，TP1/Message Queue では，問い合わせメッセージのメッセージ識別子を，アプリケーションで応答メッセージの CorrelId フィールドにコピーすることをお勧めします。

通常，グループの最初のメッセージがキューに登録されるときに，キューマネジャによってグループ識別子が生成されます。MsgSeqNumber フィールドはメッセージのグループ内での位置を識別し，Offset フィールドはメッセージ内のセグメントを識別します。

一つ以上のメッセージが複数の選択基準を満たす場合には，キューの MsgDeliverySequence 属性によって，メッセージが FIFO で選択されるのか優先度の順序で選択されるのかが決定されます。メッセージが同じ優先度を持つときは FIFO で選択されます。詳細については，「10.3　メッセージがキューから取り出される順序」を参照してください。

3.9　転送に失敗したメッセージ

キューマネジャがメッセージをキューに登録できないときに，ユーザは次に示す対応を選択できます。

- キューに再度メッセージを登録します。
- メッセージが送信側に返されるよう要求します。
- デッドレターキューにメッセージを登録します。

詳細については，「5.　プログラムエラーの処理」を参照してください。

3.10　ロールバックされたメッセージ

トランザクション制御下でキューからのメッセージを処理するとき，トランザクションは一つ以上のメッセージで構成されます。ロールバックが発生する場合，キューから取り出されたメッセージはキューに戻され，ほかのトランザクションで再度処理できるようになります。特定メッセージの処理で問題が発生する場合，再度，ロールバックされます。これによって処理がループします。キューに登録されたメッセージは削除されます。

このようなループに入ったメッセージをアプリケーションで検出するには，MQMD 構造体の BackoutCount フィールドを調べてください。アプリケーションで状態を修正したり，オペレータに警告を発行したりできます。

TP1/Message Queue でサポートする HardenGetBackout 属性は MQQA_BACKOUT_NOT_HARDENED です。

メッセージのコミットとロールバックについては，「12.　コミットとロールバック」を参照してください。

3.11　応答キューとキューマネジャ

自分が送信したメッセージに対して，次に示すメッセージを受信することがあります。

- 問い合わせメッセージに対する応答メッセージ
- 予期しないイベントまたは保持時間経過についての報告メッセージ
- メッセージ到着確認（COA）またはメッセージ配布確認（COD）イベントについての報告メッセージ
- 肯定動作の通知（PAN）または否定動作の通知（NAN）イベントについての報告メッセージ

MQMD 構造体の ReplyToQ フィールドを使用して，応答メッセージと報告メッセージを受信したいキューの名前を指定します。ReplyToQMgr フィールドには応答キューを保持するキューマネジャの名前を指定してください。

ReplyToQMgr フィールドを空白のまま放置すると，キューマネジャがメッセージ記述子の次に示すフィールドの内容を設定します。

ReplyToQ
: ReplyToQ がリモートキューのローカル定義の場合，ReplyToQ フィールドにはリモートキューの名前が設定されます。それ以外の場合では，このフィールドは設定されません。

ReplyToQMgr
: ReplyToQMgr がリモートキューのローカル定義の場合，ReplyToQMgr フィールドにはリモートキューを保持するキューマネジャの名前が設定されます。メッセージが転送失敗後に破棄されない場合には，リモートキューマネジャはデッドレターキューにメッセージを登録します。

注意
: キューマネジャがメッセージの転送を何回も再試行するように mqtalccha 定義コマンドで設定できます。メッセージの転送失敗時にメッセージを破棄しない場合には，リモートキューマネジャはメッセージをデッドレターキューに登録できます。

3.12　メッセージコンテキスト

メッセージコンテキスト情報によって，メッセージを取り出すアプリケーションは，メッセージの登録元についての情報を取得できます。取り出しアプリケーションでは，次に示すような処理を実行します。

- 送信側アプリケーションが適切な認証権限を持っているかどうかの確認
- 送信側アプリケーションに対応する作業についての課金
- 作業した全メッセージの清算記録

MQPUT または MQPUT1 命令を使用してメッセージをキューに登録するとき，キューマネジャがメッセージ記述子にデフォルトのコンテキスト情報を追加するように指定できます。適切な認証権限を持つアプリケーションは，特別なコンテキスト情報を追加できます。詳細については，「9.4　コンテキスト情報の制御」を参照してください。

すべてのコンテキスト情報はメッセージ記述子の八つのコンテキストフィールドにあります。情報の種類は二つのカテゴリに分類されます。識別コンテキスト情報と登録元コンテキスト情報です。

3.12.1　識別コンテキスト

識別コンテキスト情報では，最初にメッセージをキューに登録したアプリケーションのユーザを識別します。

- キューマネジャは，ユーザを識別する名前を UserIdentifier フィールドに設定します。方法は，アプリケーションが動作する環境によって異なります。
- キューマネジャは，メッセージを登録したアプリケーションから決定されたトークンまたは数字を AccountingToken フィールドに設定します。
- アプリケーションは ApplIdentityData フィールドを使用して，ユーザについて追加したい特別な情報（暗号化されたパスワードなど）を設定できます。

適切に認証されたアプリケーションは，上記のフィールドを設定できます。

キューマネジャ間でメッセージを受け渡しするアプリケーションでも，他アプリケーションがメッセージの登録元を知ることができるように，識別コンテキストを受け渡すようにしてください。

3.12.2　登録元コンテキスト

登録元コンテキスト情報には，現在キューに登録されているメッセージの登録元アプリケーションを記述します。メッセージ記述子には登録元情報として次に示すフィールドがあります。

- PutApplType は，メッセージを登録したアプリケーションの種類（OpenTP1 アプリケーションなど）です。

- PutApplName は，メッセージを登録したアプリケーション（ジョブやトランザクションなど）の名前です。
- PutDate は，メッセージがキューに登録された日付です。
- PutTime は，メッセージがキューに登録された時刻です。
- ApplOriginData は，アプリケーションがメッセージの登録元について保持する特別な情報です。例えば，識別データを信用するかどうかを指示するため，適切に認証されたアプリケーションによって設定されます。

登録元コンテキスト情報は，通常，キューマネジャによって設定されます。PutDate および PutTime フィールドには，グリニッジ標準時（GMT）が使用されます。

必要な権限を持つアプリケーションは，独自のコンテキストを設定できます。これによって，一人のユーザが各システムに異なるユーザ ID を持ち，登録したメッセージを処理する場合にも，アカウント情報が保持されます。

4 オブジェクト

この章では，オブジェクトについて説明します。

4.1　オブジェクトの種類

TP1/Message Queue には，次に示すオブジェクトがあります。

- キューマネジャ
- キュー
- プロセス定義
- チャネル

キューマネジャはこれらのオブジェクトの属性を定義します。これらの属性の値によって，TP1/Message Queue がオブジェクトを処理するときの動作は異なります。アプリケーションからは，MQI 命令を使用してこれらのオブジェクトを操作できます。アプリケーションから操作するとき，各オブジェクトはオブジェクト識別子（MQOD 構造体）によって識別されます。

コマンドを使用してオブジェクトを作成，変更，または削除するとき，操作に必要な権限をユーザが持っているか，キューマネジャは確認します。同様に，アプリケーションがオブジェクトをオープンするために MQOPEN 命令を使用するとき，オブジェクトへのアクセスを許可する前に，アプリケーションが必要な権限を持つかキューマネジャは確認します。確認はオープンされるオブジェクトの名前に対して実行されます。

4.2　キューマネジャ

キューマネジャはアプリケーションにサービスを提供します。アプリケーションはキューマネジャのサービスを使用する前に，キューマネジャへ接続しなくてはなりません。

キューはキューマネジャに属しますが，アプリケーションからは異なるキューマネジャに属するキューにメッセージを送信できます。

4.2.1　キューマネジャの属性

各キューマネジャには属性が関連づけられ，キューマネジャの動作を定義します。幾つかの属性はキューマネジャの作成時に決定されます。また，MQINQ 命令を使用すると，すべての属性について照会できます。

決定される属性を次に示します。

- キューマネジャの名前
- キューマネジャが処理するメッセージに付加される優先度の最大値
- MQI 命令を処理するときにキューマネジャが文字列に使用する文字セット識別子
- キューマネジャが処理できるメッセージの最大長
- アプリケーションがメッセージを登録したり取り出したりするときの，同期点のサポート

4.2.2　キューマネジャと負荷管理

同じキューについて複数の定義を持つキューマネジャクラスタを設定できます。特定のキューに対するメッセージは，キューのインスタンスを保持する任意のキューマネジャによって処理されます。ワークロード管理アルゴリズムはどのキューマネジャがメッセージを処理するかを決定し，キューマネジャ間で負荷を分散させます。

4.3 キュー

キューは，アプリケーションがメッセージを登録したり取り出したりできる名前付きオブジェクトです。メッセージはキューに蓄積されるので，登録アプリケーションがメッセージに対する応答を待つ間には，任意の処理ができます。アプリケーションは MQI 命令を使用して，キューにアクセスします。

メッセージをキューに登録する前には，キューをあらかじめ作成しなければなりません。キューはキューマネジャによって保持され，キューマネジャは複数のキューを保持できます。しかし，キューはキューマネジャ内でユニークな名前にしてください。

キューはキューマネジャによって管理されます。しかし，そのことはアプリケーションからは意識されません。

キューを作成するには，コマンド（mqamkque コマンド）を使用できます。また，TP1/Message Queue ではアプリケーションから動的にローカルキューを作成することもできます。

キューを使用する前に，実行したいことを設定してキューをオープンしてください。例えば，次に示す目的でオープンできます。

- メッセージの検索（削除はしません）
- メッセージの取り出し（アクセスを他アプリケーションと共有したり，排他したりできます）
- キューへのメッセージの登録
- キューの属性の照会
- キューの属性の設定

4.3.1 キュータイプ

アプリケーションで使用するために，TP1/Message Queue がサポートするキュータイプを次に示します。

ローカルキューとリモートキュー

アプリケーションの接続するキューマネジャによってキューが保持される場合は，アプリケーションにとってはローカルキューです。異なるキューマネジャによってキューが保持される場合には，リモートキューです。両者の大きな違いは，ユーザがメッセージを取り出せるのはローカルキューからだけであるということです。メッセージの登録は両者に対して実行できます。

ローカルキューを定義するときに作成されたキュー定義オブジェクトは，キューに登録された物理メッセージと同様に，キューの定義情報を保持します。リモートキューを定義するときに作成されたキュー定義オブジェクトは，ユーザがメッセージを送信したいキューをローカルキューマネジャが検出するのに必要な情報を保持します。リモートキューのすべての属性は，そのキューを保持するキューマネジャによって保持されます。そのキューマネジャにとってローカルキューだからです。

別名キュー

アプリケーションにとって別名キューはキューのように見えますが，実際にはほかのキューにアクセスするために使用できるオブジェクトです。したがって，複数のアプリケーションが異なる名前を使ってアクセスしながら，同じキューに対しての処理ができます。

モデルキューと動的キュー

モデルキューはキュー定義のひな形であり，動的ローカルキューを作成したいときにだけ使用します。この場合，MQA サービス定義ファイルでモデルキューを定義してください。

ひな形として属性を使用したいモデルキューの名前を指定して，アプリケーションから動的にローカルキューを作成できます。その後，必要に応じて，新しいキューの属性を変更できます。ただし，定義タイプは変更できません。

例えば，永続キューが必要なときは，永続キューとして定義タイプが設定されたモデルキューを選択しなければなりません。会話型アプリケーションでは照会に対する応答を動的キューで保持することがあります。応答を処理したあとではキューを保持する必要がないからです。

クラスタキュー

クラスタキューはクラスタキューマネジャによって保持されるキューであり，クラスタ内のキューマネジャから利用できます。

クラスタキューマネジャは mqamkque コマンドにクラスタ名を指定したキューを使用できます。クラスタ名を指定すると，クラスタ内の他キューマネジャにキューの存在が通知されます。クラスタ内にある他キューマネジャは，対応するローカルキューのリモート定義がなくても，クラスタキューにメッセージを登録できます。クラスタキューは一つ以上のクラスタに通知されます。クラスタの詳細については，マニュアル「TP1/Message Queue 使用の手引」を参照してください。

4.3.2 ローカルキューのタイプ

各キューマネジャは，特定の目的に使用する次に示すタイプのローカルキューを保持します。

転送キュー

転送キューは，リモートキューあてのメッセージを格納するローカルキューです。MQT サーバのチャネルが確立されるときに，メッセージはあて先キューに転送されます。

イニシエーションキュー

イニシエーションキューは，ローカルキューで発生した特定の条件（10 件以上のメッセージ到着など）に対してアプリケーションを自動的に開始する目的で，キューマネジャがメッセージを登録するローカルキューです。

デッドレターキュー

デッドレターキューは，キューマネジャとアプリケーションが転送できなかったメッセージを登録するローカルキューです。このキューに到着したメッセージを処理することを検討してください。

以降で詳細について説明します。

4.3.3　キューの属性

キューを定義するとき属性を指定できます。ただし，あとからは変更できない属性（キュータイプなど）もあります。変更可能なキュー属性の変更方法を次に示します。

- コマンドでの変更
  キューのテキスト記述子などがあります。
- MQSET 命令を使用するアプリケーションによる変更
  キューへのメッセージ登録の可否などがあります。

なお，MQINQ 命令を使用するとすべての属性を照会できます。

複数のキューで共通する属性を次に示します。

- QName
  キューの名前
- QType
  キュータイプ
- QDesc
  キュー記述子
- InhibitGet
  アプリケーションによるキューからのメッセージ取り出しの可否。ただし，リモートキューからはメッセージを取り出せません。
- InhibitPut
  アプリケーションによるキューへのメッセージ登録の可否
- DefPriority
  キューに登録するメッセージのデフォルトの優先度
- DefPersistence
  キューに登録するメッセージのデフォルトの永続性
- Scope
  セルディレクトリの範囲でキューのエントリがあるかどうかを指定します。

4.3.4　リモートキュー

アプリケーションにとって，自分が接続するキューマネジャ以外のキューマネジャによって保持されるキューは，リモートキューです。チャネルのコネクションが確立されると，アプリケーションが登録したメッセージはリモートキューに送信されます。アプリケーションがリモートキューからメッセージを取り出すことはできません。

リモートキューをオープンするには，次に示す名前を指定します。

- リモートキューを定義するローカル定義の名前

  リモートキューのローカル定義は MQA サービス定義の mqaremque 定義コマンドで指定します。詳細については，マニュアル「TP1/Message Queue 使用の手引」を参照してください。

  アプリケーションからはローカルキューをオープンすることと同じです。アプリケーションからは，キューがローカルなのかリモートなのかを意識しません。
- リモートキューマネジャの名前，およびそのキューマネジャが保持するキューの名前

リモートキューのローカル定義は，キューに共通の属性に加えて，三つの属性を持ちます。それらは，RemoteQName（保持するキューマネジャが認識する名前），RemoteQMgrName（保持するキューマネジャの名前），および XmitQName（ほかのキューマネジャにメッセージを転送するときに使用するローカルの転送キューの名前）です。

リモートキューのオープンの詳細については，「8.　オブジェクトのオープンとクローズ」を参照してください。

リモートキューのローカル定義に対して MQINQ 命令を使用する場合，キューマネジャはローカル定義の属性だけを返します。返される情報は，リモートキューの名前，リモートキューマネジャの名前，および転送キューの名前です。リモートシステムにある該当するローカルキューの属性ではありません。

4.3.5　別名キュー

別名キューは，ほかのキューにアクセスするために使用できるオブジェクトです。解決された別名キュー（ベースキュー）は，ローカルキューまたはリモートキューのローカル定義になります。どちらの場合でも定義済みキューまたは動的キューになります。

注意

　別名キューを異なる別名キューには解決できません。

別名キューの使用例としては，別名キューが解決されるベースキューと別名キューに異なるアクセス権限をシステム管理者が与えることがあります。つまり，アプリケーションまたはユーザが別名キューを使用することは認証されるが，ベースキューを使用することは認証されないようにできます。

反対に，別名キューへの登録操作は禁止するが，ベースキューへの登録操作は許可するように設定することもできます。

アプリケーションによっては，別名キューを使用するとシステム管理者がアプリケーションを変更しなくても，容易に別名キューオブジェクトの定義を変更できることがあります。

アプリケーションが名前を使用するときに，TP1/Message Queue は別名キューに対して認証確認をします。別名キューが解決されるベースキューへのアクセスがアプリケーションに許可されているかは確認しません。こうして，アプリケーションは別名キューの名前でのアクセスを認証されるが，解決されるキューの名前でのアクセスはできなくなります。

別名キューの InhibitGet および InhibitPut 属性は別名キューの名前に属します。例えば，別名キュー ALIAS1 がベースキュー BASE に解決される場合，ALIAS1 への禁止は ALIAS1 だけに影響し，BASE は禁止されません。しかし，BASE への禁止は ALIAS1 にも影響します。

DefPriority および DefPersistence 属性も別名キューに属します。同じベースキューに由来する異なる別名キューに，異なるデフォルト優先度を設定できます。また，別名キューを使用するアプリケーションを変更しなくても，これらの優先度を変更できます。

4.3.6 モデルキュー

モデルキューはキュー定義のひな形であり，ユーザが動的キューを作成するときに使用できます。MQOPEN 命令のオブジェクト記述子（MQOD 構造体）にモデルキューの名前を指定します。モデルキューの属性を使用して，キューマネジャが動的にローカルキューを作成します。

動的キューには完全な名前を指定できます。または共通部分だけを指定してキューマネジャにユニークな部分を追加させることができます。またはキューマネジャに完全な名前をユニークに設定させることができます。キューマネジャが名前を設定するときは，MQOD 構造体に登録されます。

MQPUT1 命令をモデルキューに対して直接は発行できません。しかし，モデルキューのオープンによって作成済みの動的キューには，MQPUT1 命令を発行できます。

モデルキューの属性はローカルキューの属性のサブセットです。

4.3.7 動的キュー

アプリケーションがモデルキューをオープンする MQOPEN 命令を発行するとき，キューマネジャはモデルキューと同じ属性でローカルキューのインスタンスを動的に作成します。モデルキューの DefinitionType 属性の値に応じて，キューマネジャは一時的動的キューまたは永続的動的キューを作成します。

(1) 一時的動的キューの特徴

一時的動的キューには，次に示す特徴があります。

- 一時的動的キューは非永続メッセージだけを保持します。
- 一時的動的キューは回復できません。
- キューマネジャ開始時に一時的動的キューは削除されます。
- MQOPEN 命令を発行してキューを作成したアプリケーションが，キューをクローズしたり終了したりした場合には，一時的動的キューは削除されます。
  - キューにコミットされたメッセージがある場合，メッセージは削除されます。

- キューについてコミットされない未解決の MQGET 命令，MQPUT 命令，または MQPUT1 命令がある場合，キューは論理的に削除されます。その後クローズ処理またはアプリケーションの終了時に，命令がコミットされると物理的に削除されます。
- キューが使用中（作成中，または他アプリケーションが使用中）の場合，キューは論理的に削除されます。その後，最後にキューを使用するアプリケーションによってクローズされるときに物理的に削除されます。
- 論理的に削除されたキューにアクセス（クローズを除く）すると，理由コード MQRC_Q_DELETED で失敗します。
- キューを作成した MQOPEN 命令に対応する MQCLOSE 命令で指定された場合は，MQCO_NONE，MQCO_DELETE，および MQCO_DELETE_PURGE はすべて MQCO_NONE として扱われます。

(2) 永続的動的キューの特徴

永続的動的キューには，次に示す特徴があります。

- 永続的動的キューは永続メッセージまたは非永続メッセージを保持します。
- 永続的動的キューはシステム障害発生時に回復できます。
- アプリケーション（MQOPEN 命令を発行してキューを作成したアプリケーション以外も含む）は，MQCO_DELETE または MQCO_DELETE_PURGE オプションを使用してキューをクローズできます。
  - キューにコミット済みまたはコミット待ちのメッセージがある場合，MQCO_DELETE オプションでのクローズ要求は失敗します。MQCO_DELETE_PURGE オプションでのクローズ要求は，キューにコミット済みのメッセージがあるときだけ成功します。メッセージはクローズ処理の一部として削除されます。ただし，キューに未解決でコミット待ちの MQGET 命令，MQPUT 命令，または MQPUT1 命令があるときは失敗します。
  - 削除要求は成功したが，キューが使用中（作成中，または他アプリケーションが使用中）の場合，キューは論理的に削除され，最後にキューを使用したアプリケーションがクローズするときに物理的に削除されます。
- キューの削除を認証されていないアプリケーション（MQOPEN 命令を発行してキューを作成したアプリケーションを除く）によってクローズされる場合には永続的動的キューは削除されません。
- 永続的動的キューはローカルキューと同様に削除できます。

(3) 動的キューの用途

ユーザは動的キューを次に示すアプリケーションで使用できます。

- 終了後にキューが保持されなくてもかまわないアプリケーション
- 他アプリケーションで処理されるメッセージの応答を要求するアプリケーション
  モデルキューのオープンによって応答キューを動的に作成できます。
  クライアントアプリケーションでの処理の例を次に示します。

1. 動的キューを作成します。
2. 問い合わせメッセージのメッセージ記述子の ReplyToQ フィールドに動的キューの名前を指定します。
3. サーバによって処理された要求をキューに登録します。

その後，サーバは応答キューに応答メッセージを登録します。最後に，クライアントアプリケーションは応答を処理して，削除オプションで応答キューをクローズします。

(4) 動的キューの検討項目

動的キューを使用する場合は，次に示す項目について検討してください。

- クライアントサーバモデルでは，各クライアントアプリケーションで独自の動的応答キューを作成して使用してください。動的応答キューが複数のクライアントアプリケーションで共有される場合には，削除は遅延することがあります。これは，コミット待ちの未処理メッセージがあったり，他クライアントアプリケーションが使用中であったりするからです。さらに，キューが論理的に削除された状態になることによって，後続の命令（MQCLOSE 命令を除く）でアクセスできないことがあります。
- 動的キューをアプリケーション間で共有する場合には，キューに対するすべての処理がコミットされたときに，キューを削除オプションでクローズするだけにしてください。最後にアクセスするユーザが実行することをお勧めします。これによって，キューの削除が遅延しなくなり，論理的に削除された状態であることからキューがアクセス不能になる時間を最小にします。

4.3.8 転送キュー

アプリケーションがメッセージをリモートキューに送信するとき，ローカルキューマネジャは，転送キューという特別なローカルキューにメッセージを格納します。

メッセージチャネルエージェント（MQT サーバ）は転送キューとリモートキューマネジャに関連づけられ，メッセージの転送を処理します。メッセージが転送されると，メッセージは転送キューから削除されます。

メッセージは複数のキューマネジャまたはノードを経由して最終あて先に届きます。経路の各キューマネジャには転送キューがあり，転送キューには次ノードへの転送を待つメッセージが保持されます。複数の転送キューがある特定のキューマネジャもあります。最終あて先が異なっても次のあて先が同じであるメッセージは，一つの転送キューに保持されます。同じリモートキューマネジャに複数の転送キューがあり，異なる種類のサービスに使用されていることもあります。

転送キューにメッセージが登録されたことを MQT サーバが検出し，チャネルを開始できます。MQT サーバのトリガ起動によるチャネル開始については，マニュアル「TP1/Message Queue 使用の手引」を参照してください。

4.3.9　イニシエーションキュー

イニシエーションキューは，アプリケーションキューでトリガイベントが発生したときに，キューマネジャがトリガメッセージを登録するローカルキューです。トリガイベントは 10 件以上のメッセージの到着といったイベントです。アプリケーション開発者は，これを合図として，キューを処理するアプリケーションをキューマネジャに開始させることができます。

トリガの詳細については，「13.　トリガによるアプリケーション開始」を参照してください。

4.3.10　デッドレターキュー

デッドレターキューは，キューマネジャが転送できないメッセージを登録するローカルキューです。

キューマネジャがデッドレターキューにメッセージを登録するとき，メッセージにヘッダを追加します。そこには，登録元メッセージのあて先，デッドレターキューにメッセージが登録された理由，日付と時刻などが記載されます。

転送できないメッセージについてアプリケーションが使用することもできます。

4.4 プロセス定義

アプリケーションをオペレータが介入しないで開始（トリガ機能）させるためには，アプリケーションの属性をキューマネジャに認識させなければなりません。これらの属性はプロセス定義オブジェクトとして定義されます。

ProcessName 属性はオブジェクト作成時に決定されます。その他の属性は，ユーザがコマンドで変更できます。すべての属性は MQINQ 命令で照会できます。

4.5 チャネル

チャネルは分散アプリケーションが使用する通信経路です。TP1/Message Queue がサポートするチャネルはメッセージチャネルです。一つのあて先に，キューマネジャ間でメッセージを転送します。

4.6　オブジェクトの命名規則

キュー，プロセス定義，およびチャネルに同じ名前を付けることができます。ただし，同じ種類のオブジェクトを同じ名前にはできません。また，大文字と小文字は区別されます。

TP1/Message Queue でオブジェクトの名前に使用できる文字を次に示します。

- 大文字の A～Z
- 小文字の a～z
- 数字の 0～9
- .（ピリオド）
- /（スラント）
- _（アンダスコア）
- %（パーセント）

注意

1. 先頭または中間に空白を使用できません。
2. 先頭または末尾にアンダスコアを使用できません。
3. フィールド長よりも短い名前は末尾を空白で埋められます。キューマネジャから返される短い名前は末尾を空白で埋められます。
4. 名前の構造（ピリオドまたはアンダスコアの使い方）はキューマネジャには関係ありません。

4.6.1　キュー名

キューの名前は二つの部分から構成されます。

- キューマネジャの名前
- キューマネジャから認識されるキューのローカル名

各部の長さは 48 文字です。

ローカルキューを参照する場合，キューマネジャの名前を空白文字にするか，または先頭をヌル文字にして省略できます。ただし，アプリケーションに返されるすべてのキュー名はキューマネジャの名前を含みます。

リモートキューを参照する場合，アプリケーションではキューマネジャの名前を完全なキュー名に含めてください。またはリモートキューのローカル定義が必要です。

アプリケーションがキュー名を使用するとき，キュー名はローカルキュー，別名キュー，またはリモートキューのローカル定義です。しかし，アプリケーションからは，キューからメッセージを取り出す場合を

除き，どこにキューがあるか意識する必要がありません。アプリケーションがキューオブジェクトをオープンするとき，以降の操作をどのキューに実行するか決定するために，MQOPEN 命令が名称解決機能を実行します。これによって，キューマネジャネットワーク内でのキューの場所をアプリケーションに組み込んで決定する必要がありません。

したがって，システム管理者がネットワーク内でのキューの場所と定義を変更すれば，これらのキューを使用するアプリケーションは変更不要です。

4.6.2　プロセス定義名

プロセス定義名の長さは最大 48 文字です。

4.6.3　チャネル名

チャネル名の長さは最大 20 文字です。

4.6.4　予約されるオブジェクト名

"SYSTEM."で始まる名前は，キューマネジャで定義されるオブジェクトに予約されています。

5 プログラムエラーの処理

この章では，プログラムエラーの処理について説明します。

5.1 プログラムエラーの種類

アプリケーションでの MQI 命令発行時，またはメッセージのあて先到着時にエラーが発生することがあります。

これらのエラーは次に示すとおりに分けられます。

- MQI 命令発行時にキューマネジャはエラーを返すことがあります。これをローカル検出エラーといいます。
- リモートキューにメッセージを送信する場合，MQI 命令発行時にはエラーが明らかになりません。この場合，エラーを検出したキューマネジャが，別のメッセージを登録元アプリケーションに送信することによって，エラーを報告します。これをリモート検出エラーといいます。

5.2 ローカル検出エラー

キューマネジャが即時に報告できる，通常のエラーを次に示します。

- MQI 命令の失敗（キュー満杯などが原因）
- アプリケーションが動作しているシステムの一部（キューマネジャなど）の中断
- データを含むメッセージの処理失敗

5.2.1 MQI 命令の失敗

キューマネジャは MQI 命令のコーディングにあるエラーを即時に報告します。そのとき定義済みリターンコードが使用されます。リターンコードは完了コードと理由コードに分けられます。

MQI 命令が成功したかどうかを示すため，キューマネジャは命令の完了時に完了コードを返します。完了コードには，正常完了，部分的完了，および命令失敗の三つがあります。キューマネジャは，部分的完了または命令失敗の理由を示す理由コードも返します。

各命令の完了コードおよび理由コードについては，マニュアル「TP1/Message Queue プログラム作成リファレンス」を参照してください。また，各命令で発生するすべてのリターンコードを処理するようにアプリケーションを作成してください。

5.2.2 システムの中断

アプリケーションの接続したキューマネジャがシステム障害で中断しても，検出できないことがあります。しかし，そのような中断が発生してもユーザデータが失われないようにアプリケーションを設計してください。

TP1/Message Queue では，データの一貫性を保証するために MQGET 命令，MQPUT 命令，および MQPUT1 命令でトランザクションへの参加を指定できます。

また，失いたくないデータを転送するには永続メッセージを使用できます。永続メッセージは，キューマネジャが障害から回復されるときに，キューに回復されます。

アプリケーションの動作中にキューマネジャがオペレータによって終了させられる場合には，MQGET 命令では理由コード MQRC_Q_MGR_STOPPING を受け取ります。この時点でユーザはアプリケーションを終了したり，MQDISC 命令を発行したりできます。

5.2.3　不正データを含むメッセージ

アプリケーション内でトランザクションを使用する場合，キューから取り出すメッセージを正常に処理できないときには，MQGET 命令がロールバックされます。キューマネジャはその回数をメッセージ記述子の BackoutCount フィールドに保持します。キューマネジャは関連する各メッセージのメッセージ記述子に回数を保持します。この回数はアプリケーションの効率を知るための有効な情報です。ロールバック回数が増加するメッセージは，繰り返し却下されています。理由を分析してメッセージを適切に処理するようアプリケーションを作成してください。

5.3　障害検出への報告メッセージの利用

MQI 命令の発行時には，リモートキューマネジャはメッセージ登録の失敗などのエラーを報告できません。しかし，メッセージがどのように処理されたかについての報告メッセージを送信できます。

アプリケーション内で，報告メッセージを MQPUT 命令で作成したり，報告メッセージを受信するオプションを選択したりできます。後者の場合，報告メッセージは他アプリケーションまたはキューマネジャによって送信されます。

5.3.1　報告メッセージの作成

報告メッセージは，送信されたメッセージを処理できないことを他アプリケーションに知らせます。メッセージを送信したアプリケーションで問題の報告を期待しているかどうかを検出するためには，Report フィールドを調査してください。報告メッセージが要求されていることを検出したあとは，次に示す項目を決定してください。

- 登録元メッセージすべてを含めるか，データの最初の 100 バイトだけを含めるか，または登録元メッセージを含めないか。
- 登録元メッセージをどのように処理するか。破棄するかまたはデッドレターキューに登録するか。
- MsgId および CorrelId フィールドの内容も必要か。

報告メッセージが生成された理由を示すには，Feedback フィールドを使用してください。報告メッセージはアプリケーションの応答キューに登録してください。

5.3.2　報告メッセージの要求と受信

他アプリケーションにメッセージを送信する場合，Report フィールドを設定して要求する報告を示さないと，問題の報告を受け取れません。

キューマネジャは，常にアプリケーションの応答キューに報告メッセージを登録します。ユーザのアプリケーションでも同じ処理をすることをお勧めします。報告メッセージ機能を使用する場合，メッセージ記述子に応答キューの名前を指定してください。指定がない場合，MQPUT 命令は失敗します。

アプリケーションには，応答キューを監視し，到着したメッセージを処理するような手続きを作成してください。報告メッセージには登録元メッセージすべてが含まれたり，最初の 100 バイトが含まれたり，何も含まれていない可能性があることを考慮してください。

キューマネジャは Feedback フィールドを設定します。これは，あて先キューがないといったエラーの理由を示すためです。アプリケーションでも同様に設定してください。

5.4　リモート検出エラー

ローカルキューマネジャが MQI 命令でエラーを検出しないで処理した場合でも，リモートキューにメッセージを送信するとき，その他の要素がリモートキューマネジャでのメッセージ処理に影響することがあります。例えば，あて先キューが満杯であったり，あて先キューがなかったりすることがあります。あて先キューまでの経路で，中間のキューマネジャによってメッセージが処理される場合には，どこかのキューマネジャでエラーが検出されます。

5.4.1　メッセージ転送の問題

MQPUT 命令が失敗したときは，送信側に返すか，またはデッドレターキューに登録するかを選択してください。

(1)　送信側へのメッセージの返却

登録元メッセージのすべてを含む報告メッセージの作成を要求することによって，送信側にメッセージを返却できます。詳細については，「3.3.4　報告メッセージ」を参照してください。

5.4.2　デッドレターキューの使用

キューマネジャがメッセージを転送できないとき，キューマネジャはデッドレターキューにメッセージを登録します。このキューはキューマネジャの開始時に定義してください。

キューマネジャと同様にアプリケーションからデッドレターキューを使用できます。キューマネジャオブジェクトを MQOPEN 命令でオープンし，DeadLetterQName 属性を MQINQ 命令で照会することによって，デッドレターキューの名前を調べることができます。

キューマネジャがメッセージをこのキューに登録するとき，メッセージにヘッダを追加します。その形式は MQDLH 構造体（デッドレターヘッダ）で定義されます。MQDLH 構造体については，マニュアル「TP1/Message Queue プログラム作成リファレンス」を参照してください。このヘッダにはあて先キューの名前やメッセージがデッドレターキューに登録された理由が記載されます。目的のキューにメッセージを登録する前には，問題を解決してヘッダを削除してください。MQDLH 構造体が含まれることを示すために，キューマネジャはメッセージ記述子の Format フィールドを変更します。

MQDLH 構造体

デッドレターキューにメッセージを登録するときには，すべてのメッセージに MQDLH 構造体を追加することをお勧めします。

ヘッダをメッセージに追加することによって，デッドレターキューへ登録するにはメッセージが長過ぎてしまうことがあります。したがって，少なくとも MQ_MSG_HEADER_LENGTH の値の分だけは，デッドレターキューに登録可能なメッセージ長よりもメッセージが短くなるように注意してください。キュー

に登録可能なメッセージの最大長は，キューの MaxMsgLength 属性の値によって決定されます。デッドレターキューについては，この属性がキューマネジャの許容する最大長になるようにしてください。アプリケーションがメッセージを転送できない場合で，メッセージがデッドレターキューへの登録には長過ぎるときには，MQDLH 構造体の解説に従って処理してください。

デッドレターキューを監視して，到着したメッセージが処理されるように独自のアプリケーションを用意してください。アプリケーションはトリガで動作させたり，一定間隔で動作させたりします。

データ変換が必要な場合には，MQGET 命令に MQGMO_CONVERT オプションを使用するときに，キューマネジャがヘッダ情報を変換します。メッセージを登録するプロセスが MCA の場合には，ヘッダのあとに登録元メッセージのすべてのテキストが続きます。

ただし，TP1/Message Queue では，MCA からデッドレターキューに登録するメッセージが長過ぎる場合，MQRC_MSG_TOO_BIG_FOR_Q で失敗します。

(1) デッドレターキューの処理

デッドレターキューの処理はローカルシステムの要件によって異なります。計画時には次に示す項目を参考にして検討してください。

- MQMD 構造体の Format フィールドが MQFMT_DEAD_LETTER_HEADER であることから，メッセージにデッドレターキューヘッダがあるとわかります。
- MCA がメッセージをデッドレターキューに登録すると，PutApplType フィールドは MQAT_QMGR になり，PutApplName フィールドはキューマネジャ名称の先頭 28 バイトになります。
- メッセージがデッドレターキューに登録された理由は，デッドレターキューヘッダの Reason フィールドにあります。
- デッドレターキューヘッダには，あて先キュー名とキューマネジャ名の詳細があります。
- メッセージをあて先キューに登録する前に回復しなければいけないフィールドは，デッドレターキューヘッダにあります。それらを次に示します。
  1. Encoding
  2. CodedCharSetId
  3. Format
- 上記の三つのフィールドを除くと，メッセージ記述子は登録元アプリケーションによる登録時と同じ状態です。

デッドレターキューアプリケーションでは，次に示す項目を一つ以上実行してください。

- Reason フィールドを調べてください。次に示す理由の場合にはメッセージは MCA によって登録されたものです。
  - メッセージがチャネルの最大長よりも長大
    理由コードは MQRC_MSG_TOO_BIG_FOR_CHANNEL

- メッセージのあて先キューへの登録不可
  理由コードは MQPUT 命令で返される MQRC_*
- UOC による要求
  UOC によって返された理由コード。TP1/Message Queue では UOC によって返される値は MQFB_STOPPED_BY_MSG_EXIT だけです。
- 配布リストを転送できない（自チャネルのバッファ方式がセグメント方式のとき，または相手チャネルが配布リストを受信できないとき）
  理由コードは MQFB_XMIT_Q_MSG_ERROR
- データ変換後のメッセージがチャネルの最大サイズよりも長大
  理由コードは MQRC_CONVERTED_MSG_TOO_BIG
- データ変換に失敗
  理由コードは次に示すどれか
  MQRC_FORMAT_ERROR
  MQRC_NOT_CONVERTED
  MQRC_SOURCE_CCSID_ERROR
  MQRC_TARGET_CCSID_ERROR
  MQFB_XMIT_Q_MSG_ERROR
- MQXQH 構造体が不正
  MQFB_XMIT_Q_MSG_ERROR

- 可能な場合は，メッセージを目的のあて先に転送してください。
- デッドレターキューへの登録の原因はわかったが即時に修正できないときは，破棄する前に一定期間メッセージを保持してください。
- 原因が分かればシステム管理者に問題の修正を指示してください。
- 破損などで処理できないメッセージは破棄してください。

デッドレターキューから回復したメッセージを処理する方法は二つあります。

- メッセージがローカルキューあての場合には，次に示す項目を実行してください。
  - アプリケーションデータを取り出すのに必要なコード変換の実行
  - ローカル機能の場合でのコード変換の実行
  - メッセージ記述子の内容を回復し，変換後のメッセージをローカルキューに登録
- メッセージがリモートキューあての場合には，メッセージをキューに登録してください。

6 MQI の概要

この章では，MQI の概要について説明します。

ここで説明する MQI 命令，構造体，基本データタイプ，およびリターンコードの詳細については，マニュアル「TP1/Message Queue プログラム作成リファレンス」を参照してください。

6.1 MQI の特徴

MQI は次に示す要素から構成されます。

- MQI 命令
  アプリケーションがキューマネジャとその機能にアクセスするための命令です。
- 構造体
  キューマネジャとデータを受け渡しするためにアプリケーションが使用します。
- 基本データタイプ
  アプリケーションがキューマネジャと受け渡しするデータタイプです。

また，TP1/Message Queue は次に示すファイルを提供します。

- インクルードファイル
  定数の値を定義するインクルードファイルです。
- ライブラリファイル
  アプリケーションをリンクするライブラリファイルです。
- サンプルプログラムファイル
  MQI の使用方法のサンプルファイルです。

6.1.1 MQI 命令

TP1/Message Queue が提供する MQI 命令は次に示すとおりに分類できます。

MQCONN 命令と MQDISC 命令
　これらの命令を使用して，キューマネジャに接続したりキューマネジャから切り離したりします。

MQOPEN 命令と MQCLOSE 命令
　これらの命令を使用して，キューなどのオブジェクトをオープンしたりクローズしたりします。

MQPUT 命令と MQPUT1 命令
　これらの命令を使用して，キューにメッセージを登録します。

MQGET 命令
　この命令を使用して，キューのメッセージを検索したり削除したりします。

MQINQ 命令
　この命令を使用して，オブジェクトの属性を照会します。

MQSET 命令
　この命令を使用して，キューの属性を設定します。キュー以外のオブジェクトの属性は設定できません。

6.1.2　同期点命令

TP1/Message Queue では，トランザクションの同期点を取得するために，OpenTP1 が提供する命令を使用します。dc_trn_chained_commit()や dc_trn_chained_rollback()などのトランザクション制御命令を使用してください。

6.1.3　構造体

MQI 命令で使用する構造体です。プログラミング言語ごとのデータ定義ファイルで提供されます。

6.1.4　基本データタイプ

サポートするプログラミング言語に対して，MQI は基本データタイプを提供します。詳細については，マニュアル「TP1/Message Queue プログラム作成リファレンス」を参照してください。

6.1.5　データ定義

TP1/Message Queue が提供するデータ定義ファイルには，次に示す項目が定義されています。

- すべての定数とリターンコード
- 構造体とデータタイプ
- 構造体を初期化するための定数定義
- 各 MQI 命令のプロトタイプ宣言（C 言語の場合）

6.2　すべての MQI 命令に共通するパラメタ

すべての命令に共通するパラメタとして，ハンドルとリターンコードがあります。

6.2.1　コネクションハンドルとオブジェクトハンドル

キューマネジャと通信するアプリケーションでは，キューマネジャを区別するユニークな識別子が必要です。この識別子をコネクションハンドルといいます。コネクションハンドルは，アプリケーションがキューマネジャに接続するときに MQCONN 命令から返されます。アプリケーションでほかの命令を使用するときには，コネクションハンドルを入力パラメタとして指定します。

オブジェクトを使用するアプリケーションでは，オブジェクトを区別するユニークな識別子が必要です。この識別子をオブジェクトハンドルといいます。オブジェクトハンドルは，アプリケーションがオブジェクトをオープンするとき MQOPEN 命令から返されます。アプリケーションで後続の MQPUT 命令，MQGET 命令，MQINQ 命令，MQSET 命令，または MQCLOSE 命令を使用するときには，オブジェクトハンドルを入力パラメタとして指定します。

アプリケーションとキューマネジャの接続の手順およびハンドルの有効範囲については，マニュアル「TP1/Message Queue 使用の手引」を参照してください。

6.2.2　リターンコード

各命令では完了コードおよび理由コードが出力パラメタとして返されます。これらをまとめてリターンコードといいます。

命令の成功と失敗を示すために，各命令は完了時に完了コードを返します。完了コードは，通常は，MQCC_OK または MQCC_FAILED です。それぞれ成功と失敗を表します。中間状態を返す命令もあり，部分的完了を表す MQCC_WARNING が返されます。

各命令は，失敗または部分的完了の理由を示す理由コードも返します。複数の理由コードがあり，キューの満杯，キューからの取り出し操作禁止，キューマネジャにキューの定義がないといった状況を表します。アプリケーションは理由コードを使用して対応を決定できます。例えば，アプリケーションのユーザにデータ入力の変更を指示してから命令を再発行したり，またはユーザにエラーメッセージを返したりできます。

完了コードが MQCC_OK のとき，理由コードは常に MQRC_NONE です。

6.3 バッファの指定

キューマネジャは必要なときだけバッファを参照します。命令のバッファが不要なときや長さが 0 のときは，ユーザはバッファにヌルポインタを使用できます。

必要なバッファの長さを指定するときは，データ長を常に指定してください。

命令からの出力（MQGET 命令のメッセージデータや MQINQ 命令で照会した属性の値など）を保持するためにバッファを使用する場合，指定したバッファが無効な場合や書き込みできないときに，キューマネジャは理由コードを返そうとします。ただし，理由コードを返せないこともあります。

注意

MQGET 命令および MQINQ 命令では，アプリケーションで動的または静的に確保した領域に，情報を返却する場合があります。

そのため，実際に確保した領域の大きさが MQGET 命令の BufferLength パラメタ，MQINQ 命令の IntAttrCount パラメタ，および MQINQ 命令の CharAttrs パラメタに指定された値より小さい場合，領域破壊など予期しない事象が発生するので動作は保証されません。

6.4　プログラミング言語の検討項目

TP1/Message Queue は次に示すプログラミング言語をサポートします。

- C 言語
- C++言語
- COBOL 言語

6.4.1　C 言語または C++言語のコーディング

C 言語または C++言語でアプリケーションを記述するときには，次に示す項目について検討してください。

(1)　MQI 命令のパラメタ

入力専用であり，かつ MQHCONN，MQHOBJ，または MQLONG 型のパラメタは値を渡します。その他のパラメタでは，パラメタのアドレスを渡します。

アドレスを渡すパラメタについては，関数発行のたびに渡す必要はありません。パラメタが不要な場合には，パラメタデータのアドレスの代わりにヌルポインタを関数パラメタとして渡せます。該当するパラメタが何であるかについては，各命令の説明を参照してください。

関数の値としてパラメタが返されることはありません。C 言語または C++言語ではすべての関数が void を返します。

(2)　未定義データタイプのパラメタ

MQGET 命令，MQPUT 命令，および MQPUT1 命令は，未定義のデータタイプである Buffer パラメタを一つ持ちます。このパラメタはアプリケーションのメッセージデータを送受信するのに使用されます。

このパラメタは MQBYTE 型の配列で定義できます。しかし，通常はメッセージ内のデータ構成を表す構造体で宣言する方が便利です。関数パラメタは void ポインタで宣言されているので，関数発行時にどのデータタイプのアドレスでも指定できます。

(3)　データタイプ

すべてのデータタイプは typedef ステートメントで定義されます。各データタイプには，対応するポインタデータタイプも定義されます。ポインタデータタイプの名前は，基本データタイプまたは構造体データタイプの名前の先頭にポインタを表す"P"を付加した名前です。ポインタの属性は MQPOINTER マクロ変数で定義されます。マクロ変数の値は環境によって異なります。ポインタデータタイプの宣言について次に示します。

```
#define MQPOINTER                    /* 環境依存 */
...
```

```
typedef MQLONG MQPOINTER PMQLONG;  /* MQLONGのポインタ */
typedef MQMD   MQPOINTER PMQMD;    /* MQMDのポインタ */
```

(4) 2進文字列の操作

2進文字列はMQBYTEnデータタイプとして宣言されます。このデータタイプをコピー，比較，および設定する場合には，memcpy()，memcmp()，およびmemset()を使用してください。

```
#include <string.h>
#include "cmqc.h"

MQMD MyMsgDesc;

memcpy(MyMsgDesc.MsgId,          /* MsgIdフィールドにヌルを設定 */
       MQMI_NONE,                /* 名前付き定数を使用する場合 */
       sizeof(MyMsgDesc.MsgId));

memcpy(MyMsgDesc.CorrelId,       /* CorrelIdフィールドにヌルを設定 */
       0x00,                     /* 違う方法 */
       sizeof(MQBYTE24));
```

文字列関数strcpy()，strcmp()，strncpy()，およびstrncmp()は使用しないでください。MQBYTE24で宣言されているデータを正しく処理できません。

(5) 文字列の操作

キューマネジャが文字データをアプリケーションに返すときは，定義された長さになるように文字データを空白で埋めます。キューマネジャがヌル文字で終わる文字列を返すことはありません。しかし，ユーザ入力で使用することは可能です。そのため，そのような文字列をコピー，比較，または結合するときには文字列関数strncpy()，strncmp()，またはstrncat()を使用してください。

文字列の終わりにヌルが必要な文字列関数（strcpy()，strcmp()，またはstrcat()）は使用しないでください。また，文字列の長さを決定するのにstrlen()は使用しないでください。代わりにsizeof()を使用してください。

(6) 構造体の初期値

インクルードファイルcmqc.hは，構造体のインスタンスが宣言されたときの初期値を設定するマクロ変数を定義します。これらのマクロ変数はMQxxx_DEFAULTという形式の名前です。MQxxxは構造体の名前です。例えば，次に示すとおり使用します。

```
MQMD MyMsgDesc = {MQMD_DEFAULT};
MQMD MyPutOpts = {MQPMO_DEFAULT};
```

MQIで有効値を定義する文字フィールド（MQMD構造体のStrucId，Formatフィールドなど）もあります。有効値として，二つのマクロ変数が提供されます。

- ヌルを除くと，フィールドに定義される長さと一致するような文字列の値を定義するマクロ変数。例を次に示します。

```
#define MQMD_STRUC_ID "MD  "
#define MQFMT_STRING  "MQSTR   "
```

  この形式を memcpy()および memcmp()で使用してください。

- 文字の配列として値を定義するマクロ変数。名前の末尾は"_ARRAY"です。例を次に示します。

```
#define MQMD_STRUC_ID_ARRAY 'M', 'D', ' ', ' '
#define MQFMT_STRING_ARRAY 'M', 'Q', 'S', 'T', 'R', ' ', ' ', ' '
```

  MQMD_DEFAULT マクロ変数の提供する値とは異なる値で構造体のインスタンスを宣言するときには，この形式を使用してフィールドを初期化してください。

(7) 動的構造体の初期値

複数の構造体のインスタンスが必要な場合には，通常は，calloc()または malloc()を使用して動的に取得した領域にインスタンスが作成されます。このような構造体のフィールドを初期化するには，次に示す方法をお勧めします。

1. 適切な MQxxx_DEFAULT マクロ変数を使用して初期化して，構造体のインスタンスを宣言します。このインスタンスが他インスタンスのモデルとなります。

```
MQMD ModelMsgDesc = {MQMD_DEFAULT};
                                  /* モデルインスタンスの宣言 */
```

   必要に応じて宣言に static または auto を使用して，モデルインスタンスに静的または動的なスコープを設定することもできます。

2. calloc()または malloc()を使用して，構造体の動的インスタンスの領域を取得します。

```
PMQMD InstancePtr;
InstancePtr = malloc(sizeof(MQMD));
                                  /* 動的インスタンスの領域を取得 */
```

3. memcpy()を使用してモデルインスタンスを動的インスタンスにコピーします。

```
memcpy(InstancePtr, &ModelMsgDesc, sizeof(MQMD));
                                  /* 動的インスタンスの初期化 */
```

(8) C++コンパイラの使用

C++コンパイラの使用時だけにインクルードされるステートメントが，ヘッダファイルにあります。

```
#ifdef __cplusplus
extern "C" {
#endif

/* ヘッダファイルの残り */
```

```
#ifdef __cplusplus
 }
#endif
```

6.4.2 COBOL 言語のコーディング

COBOL 言語でアプリケーションを記述するときには，次に示す項目について検討してください。

(1) 名前付き定数

このマニュアルでは，定数の名前の一部にアンダスコアを使用して表記しています。COBOL 言語ではアンダスコアの代わりにハイフンを使用してください。

文字列の値の定数はデリミタとしてアポストロフィを使用します。コンパイラで処理できるように APOST オプションを使用してください。

コピーファイル CMQV はレベル 10 の項目として名前付き定数を定義します。定数を使用するには，レベル 01 の項目を明示的に宣言し，定数の宣言で COPY ステートメントを使用してください。

```
WORKING-STORAGE SECTION.
01  MQM-CONSTANTS.
    COPY CMQV.
```

しかし，この方法では参照がないときにもアプリケーションの領域を定数が占めることになります。同じ動作単位内で異なるアプリケーションに定数がある場合には，複数の定数のコピーがあることになります。そのため，多くの領域が使用されてしまいます。この状況を回避するには，GLOBAL 節をレベル 01 の宣言に追加してください。

```
* 定数を保持するグローバル構造体の宣言
 01  MQM-CONSTANTS GLOBAL.
     COPY CMQV.
```

GLOBAL 節でアロケートされるのは，動作単位内で 1 組の定数だけです。これらの定数は，レベル 01 宣言のあるアプリケーションでなくても，動作単位内の任意のアプリケーションで参照できます。

7 キューマネジャの接続と切り離し

この章では，キューマネジャの接続と切り離しについて説明します。

7.1 MQCONN 命令によるキューマネジャの接続

ユーザはキューマネジャを指定して接続したり，デフォルトのキューマネジャに接続したりできます。ただし，接続するキューマネジャは，アプリケーションと同じシステムで動作する必要があります。

TP1/Message Queue ではアプリケーションと同じシステムで動作するキューマネジャは一つだけです。

デフォルトのキューマネジャに接続するには，MQCONN 命令に空白またはヌル文字を先頭とする名前を指定してください。

UNIX システム上で使用する場合は，キューマネジャに接続するアプリケーションは認証されていなければなりません。つまり，OpenTP1 グループのグループ ID が必要です。

MQCONN 命令からの出力を次に示します。

- コネクションハンドル
- 完了コード
- 理由コード

以降の MQI 命令ではコネクションハンドルを使用してください。

アプリケーションがすでにキューマネジャに接続しているという理由コードの場合には，返されているコネクションハンドルは最初の接続時に返されたのと同じです。アプリケーションを接続したままにしたい場合には，MQDISC 命令の発行は不要です。

コネクションハンドルのスコープはオブジェクトハンドルと同じです。

キューマネジャが休止状態または終了状態の場合は，MQCONN 命令は失敗します。

7.1.1 MQCONN 命令のスコープ

TP1/Message Queue では，MQCONN 命令のスコープは，プロセス単位です。各プロセスは異なるキューマネジャに接続できます。

7.2 MQDISC 命令によるキューマネジャからの切り離し

MQCONN 命令でキューマネジャに接続したアプリケーションですべての処理が終われば，MQDISC 命令を使用して接続を切断します。

MQDISC 命令の発行後は，コネクションハンドルが無効になります。MQCONN 命令を再度発行するまでは，以降の MQI 命令は発行できません。また，MQDISC 命令を発行すると，該当するハンドルを使用してオープン中のオブジェクトについて，MQCLOSE 命令が暗黙的に発行されます。

MQDISC 命令の入力には，キューマネジャへの接続時に MQCONN 命令が返したコネクションハンドルを指定してください。

MQDISC 命令の出力は完了コードと理由コードです。コネクションハンドルには，
MQHC_UNUSABLE_HCONN の値が設定されます。

7.2.1 権限確認

MQCLOSE 命令と MQDISC 命令は権限を確認しません。通常は，接続やオブジェクトのオープンの権限を持つ業務が，切断やクローズを実行します。しかし，接続やオープンを実行した業務の権限が取り消されていても，MQCLOSE 命令と MQDISC 命令は受け付けられます。

8 オブジェクトのオープンとクローズ

この章では，オブジェクトのオープンとクローズについて説明します。

8.1　オブジェクトのオープンとクローズの概要

次に示す操作を実行するには，最初にオブジェクトをオープンしなくてはなりません。

- キューへのメッセージの登録
- キューからのメッセージの取り出し（検索と削除）
- オブジェクトの属性の設定
- オブジェクトの属性の照会

オブジェクトをオープンするには，どのオブジェクトを処理したいのかを指定して MQOPEN 命令を発行してください。ただし，1 メッセージをキューに登録して即座にキューをクローズする場合は除きます。この場合は，MQPUT1 命令を使用して，オープンの手続きを省略します。

MQOPEN 命令を使用してオブジェクトをオープンする前には，アプリケーションをキューマネジャに接続してください。

TP1/Message Queue がオープンできるオブジェクトの種類を次に示します。

- キュー
- プロセス定義
- キューマネジャ

MQOPEN 命令を使用して，すべてのオブジェクトを同じ要領でオープンできます。

同じオブジェクトを複数回オープンできます。そのたびに新しいオブジェクトハンドルを取得できます。一つのハンドルでキューのメッセージを検索したり，ほかのハンドルを使用して同じキューからメッセージを削除したりできます。この場合は，同じオブジェクトをクローズして再度オープンするようにするとリソースの消費を回避できます。また，同じキューをオープンしてメッセージの検索と削除を同時に実行できます。

さらに，配布リストを使用することによって，1 回の MQOPEN 命令で複数のオブジェクトをオープンしてから MQCLOSE 命令でクローズできます。詳細については，「9.6　配布リスト」を参照してください。

オブジェクトをオープンするときには，MQOPEN 命令に指定するオプションについて，オブジェクトをオープンする権限があるかをキューマネジャが確認します。

アプリケーションがキューマネジャから切り離されるときは，オブジェクトは自動的にクローズされます。

オープンしたオブジェクトはクローズすることをお勧めします。MQCLOSE 命令を使用して実行してください。

8.2　MQOPEN 命令によるオブジェクトのオープン

MQOPEN 命令の入力として，次に示す項目を指定してください。

- コネクションハンドル
  MQCONN 命令で返されたコネクションハンドルを使用してください。
- オープンしたいオブジェクトのオブジェクト記述子（MQOD 構造体）
- 命令の動作を制御する一つ以上のオプション

MQOPEN 命令の出力を次に示します。

- オブジェクトへのアクセスを表すオブジェクトハンドル。以降の MQI 命令の入力に指定してください。
- 変更されたオブジェクト記述子（動的キューを作成する場合）
- 完了コード
- 理由コード

8.2.1　オブジェクトハンドルのスコープ

オブジェクトハンドルのスコープは，コネクションハンドルのスコープと同じです。

TP1/Message Queue では，同じプロセス内で同じハンドルを使用できます。

8.2.2　オブジェクトの識別（MQOD 構造体）

オープンしたいオブジェクトを識別するには MQOD 構造体を使用してください。この構造体は MQOPEN 命令の入力パラメタです。動的キューを作成するために MQOPEN 命令を使用するときには，キューマネジャによって変更されます。

配布リストに MQOD 構造体を使用する方法については，「9.6　配布リスト」を参照してください。

8.2.3　名称解決

注意事項

リモートキューのローカル定義でリモートキュー名を指定する，mqaremque 定義コマンドの-r オプションを省略すると，キューマネジャの別名定義になります。

キューをオープンするとき，指定したキュー名について MQOPEN 命令は名前を解決します。これによって，キューマネジャは以降の操作を実行するキューを決定します。つまり，ユーザがオブジェクト記述子

に別名キューまたはリモートキューの名前を指定するとき，命令はそれらをローカルキューまたは転送キューの名前に解決します。

登録，検索，または設定用にキューがオープンされる場合は，ローカルキューに解決されます。ローカルキューがなければ命令は失敗します。出力専用，照会専用，または出力と照会用にオープンされる場合は，ローカルキュー以外に解決されます。MQOPEN 命令使用時のキュー名解決について，表 8-1 に示します。ユーザが ObjectQMgrName フィールドに指定した名前は，ObjectName フィールドに指定した名前よりも先に解決されることに注意してください。

この表では，キューマネジャの別名を定義するために，リモートキューのローカル定義を使用する方法も示しています。ユーザはリモートキューにメッセージを登録するときに，どの転送キューを使用するか選択できます。例えば，複数のリモートキューマネジャあてのメッセージに一つの転送キューを使用できます。

この表を使用する場合は，"MQOD 構造体の入力"列を参照して，該当する行を選択してください。同じ行の"解決後の名前"列には，どのキューマネジャのどのオブジェクトに解決されるかが示されます。

表 8-1　MQOPEN 命令使用時のキュー名解決

項番	MQOD 構造体の入力		解決後の名前		転送キュー
	ObjectQMgrName	ObjectName	ObjectQMgrName	ObjectName	
1	空白またはローカルキューマネジャ	クラスタ属性のないローカルキュー	ローカルキューマネジャ	入力値	－（ローカルキューが使用されます。）
2	空白	クラスタ属性のローカルキュー	負荷分散機能が選択したクラスタキューマネジャ，または MQPUT 命令で選択された特定のクラスタキューマネジャ	入力値	SYSTEM.CLUSTER.TARASMIT.QUEUE およびローカルキューが使用されます。
3	ローカルキューマネジャ	クラスタ属性のローカルキュー	ローカルキューマネジャ	入力値	－（ローカルキューが使用されます。）
4	空白またはローカルキューマネジャ	モデルキュー	ローカルキューマネジャ	生成された名前	－（ローカルキューが使用されます。）
5	空白またはローカルキューマネジャ	別名キュー。クラスタ属性の有無は関係ありません。	ObjectQMgrName フィールドの値はそのままで別名キュー定義オブジェクトにある BaseQName の値を ObjectName に設定して名称解決を再度実行します。	－	－

項番	MQOD 構造体の入力		解決後の名前		転送キュー
	ObjectQMgrName	ObjectName	ObjectQMgrName	ObjectName	
5	空白またはローカルキューマネジャ	別名キュー。クラスタ属性の有無は関係ありません。	別名キューには戻りません。	－	－
6	空白またはローカルキューマネジャ	リモートキューのローカル定義。クラスタ属性の有無は関係ありません。	RemoteQMgrName フィールドの値を ObjectQMgrName に，RemoteQName の値を ObjectName に設定して名称解決を再度実行します。 リモートキューには戻りません。	－	空白でない場合は XmitQName 属性の名前。それ以外の場合はリモートキュー定義オブジェクトの RemoteQMgrName
7	空白	該当するローカルオブジェクトがなく，クラスタキューがあります。	負荷分散機能が選択したクラスタキューマネジャ，または MQPUT 命令で選択された特定のクラスタキューマネジャ	入力値	SYSTEM.CLUSTER.TARASMIT.QUEUE
8	空白またはローカルキューマネジャ	該当するローカルオブジェクトがなく，クラスタキューがありません。	－	キューが見つからないエラー	－
9	ローカル転送キューの名前	解決されません。	入力値	入力値	入力値
10	キューマネジャの別名定義（RemoteQMgrName フィールドはローカルキューマネジャ）	解決されません。リモートキューです。	RemoteQMgrName フィールドの値を ObjectQMgrName に設定して名称解決を再度実行します。 リモートキューには戻りません。	入力値	空白でない場合は XmitQName 属性の名前。それ以外の場合はリモートキュー定義オブジェクトの RemoteQMgrName
11	ローカルオブジェクト以外のキューマネジャ（クラスタキューマネジャまたはキューマネジャの別名）	解決されません。	入力値，または MQPUT 命令で選択された特定のクラスタキューマネジャ	入力値	SYSTEM.CLUSTER.TARASMIT.QUEUE
12	ローカルオブジェクト以外のキューマネジャ（クラスタオブ	解決されません。	入力値	入力値	キューマネジャの DefXmitQName 属性

項番	MQOD 構造体の入力		解決後の名前		転送キュー
	ObjectQMgrName	ObjectName	ObjectQMgrName	ObjectName	
12	ジェクトはありません。）	解決されません。	入力値	入力値	キューマネジャの DefXmitQName 属性

（凡例）

－：該当しません。

注意

- BaseQName は，別名キューの定義に指定したベースキュー名です。
- RemoteQName は，リモートキューのローカル定義に指定したリモートキュー名です。
- RemoteQMgrName は，リモートキューのローカル定義に指定したリモートキューマネジャ名です。
- XmitQName は，リモートキューのローカル定義に指定した転送キュー名です。

別名キューをオープンすると，別名の解決されたベースキューもオープンされます。リモートキューをオープンすると，転送キューもオープンされます。そのため，一方がオープンされている間は，指定したキューも解決されたキューも削除できません。

解決されたキュー名と解決されたキューマネジャ名は，MQOD 構造体の ResolvedQName フィールドと ResolvedQMgrName フィールドに保存されます。

8.2.4 MQOPEN 命令オプションの使用

MQOPEN 命令の Options パラメタには，オブジェクトへのアクセスを制御するオプションを設定してください。オプションを次に示します。

- 該当するキューに登録されたすべてのメッセージが，同じインスタンスに格納されるようにキューをオープンします。
- メッセージの登録を許可するようにキューをオープンします。
- メッセージの検索を許可するようにキューをオープンします。
- メッセージの削除を許可するようにキューをオープンします。
- 属性の照会と設定を許可するようにオブジェクトをオープンします。設定できるのはキューの属性だけです。
- コンテキスト情報をメッセージに関連づけます。
- 代替ユーザ識別子を使用してキューをオープンします。

(1) クラスタキューへの MQOPEN 命令オプション

MQPUT 命令でキューに登録されたすべてのメッセージを，同じ経路で同じキューマネジャに導くには MQOPEN 命令に MQOO_BIND_ON_OPEN オプションを使用してください。MQPUT 命令時にメッセージ単位であて先を指定するには，MQOPEN 命令に MQOO_BIND_NOT_FIXED オプションを使用してください。どちらも指定しない場合は，デフォルトの MQOO_BIND_AS_Q_DEF が使用されます。この場合は，キューハンドルに使用されるバインディングはキューの DefBind 属性から取得されます。DefBind 属性には MQBND_BIND_ON_OPEN または MQBND_BIND_NOT_FIXED の値が設定されます。MQOD 構造体にローカルキューマネジャの名前を指定する場合は，クラスタキューのローカルインスタンスが選択されます。キューマネジャ名が空白の場合は，任意のインスタンスが選択されます。

(2) メッセージ登録の MQOPEN 命令オプション

メッセージを登録するためにキューをオープンするには，MQOPEN 命令に MQOO_OUTPUT オプションを使用してください。

(3) メッセージ検索の MQOPEN 命令オプション

キューのメッセージを検索するためにキューをオープンするには，MQOPEN 命令に MQOO_BROWSE オプションを使用してください。キューマネジャがキューの次メッセージを識別するために使用する検索カーソルが作成されます。

注意

1. リモートキューのメッセージは検索できません。そのため，MQOO_BROWSE オプションを使用してリモートキューをオープンできません。
2. 配布リストをオープンするときには，このオプションは指定できません。

(4) メッセージ削除の MQOPEN 命令オプション

キューからメッセージを削除するために，キューのオープンを制御するオプションは三つあります。これらのうちの一つを MQOPEN 命令に使用できます。これらのオプションは，アプリケーションがキューに排他アクセスや共用アクセスをすることを指定します。

排他アクセスでは，キューをクローズするまで，該当アプリケーションだけがメッセージを削除できます。他アプリケーションがメッセージを削除するためにキューをオープンすると，MQOPEN 命令は失敗します。共用アクセスでは，複数のアプリケーションがキューからメッセージを削除できます。

キューの定義時に指定されたアクセス方法を使用することをお勧めします。キューの定義には，Sharebility および DefInputOpenOption 属性があります。この方法を利用するには，MQOO_INPUT_AS_Q_DEF オプションを使用してください。

キュー属性と MQOPEN 命令オプションに対応するキューアクセスについて，次の表に示します。

表 8-2　キュー属性と MQOPEN 命令オプションに対応するキューアクセス

キュー属性		MQOPEN 命令オプション		
Sharebility	DefInputOpenOption	AS_Q_DEF	SHARED	EXCLUSIVE
SHAREABLE	SHARED	共用	共用	排他
SHAREABLE	EXCLUSIVE	排他	共用	排他
NOT_SHAREABLE※	SHARED※	排他	排他	排他
NOT_SHAREABLE	EXCLUSIVE	排他	排他	排他

注※

この組み合わせもキューに定義できますが，DefInputOpenOption 属性は，Sharebility 属性で上書きされます。

MQOPEN 命令の使い方を次に示します。

- 他アプリケーションがキューからメッセージを同時に削除する場合にも，アプリケーションが正常に動作できるときには，MQOO_INPUT_SHARED オプションを使用してください。表 8-2 に示すように，このオプション指定時でもキューに排他アクセスが設定されることがあります。
- 他アプリケーションが同時にキューからメッセージを削除できない場合にだけ，アプリケーションが正常に動作できるときには，MQOO_INPUT_EXCLUSIVE オプションを指定してください。

注意

1. リモートキューからはメッセージを削除できません。そのため，MQOO_INPUT_*オプションを使用してリモートキューをオープンできません。
2. 配布リストをオープンするときは，このオプションは指定できません。詳細については，「9.6　配布リスト」を参照してください。

(5)　属性の設定と照会の MQOPEN 命令オプション

属性を設定するためにキューをオープンするには，MQOO_SET オプションを使用してください。ほかの種類のオブジェクトには，属性を設定できません。

属性を照会するためにキューをオープンするには，MQOO_INQUIRE オプションを使用してください。

注意

配布リストをオープンするときは，このオプションは指定できません。

(6)　メッセージコンテキストに関連する MQOPEN 命令オプション

キューにメッセージを登録するときに，コンテキスト情報をメッセージに関連づけるには，キューのオープン時にメッセージコンテキストオプションを使用してください。

このオプションによって，メッセージの登録元ユーザに関連するコンテキスト情報とメッセージ登録元アプリケーションに関連するコンテキスト情報を区別できます。また，キューにメッセージを登録するときにコンテキスト情報を設定したり，ほかのキューハンドルから自動的にコンテキスト情報を取得したりできます。

メッセージコンテキストの詳細については，「3.12　メッセージコンテキスト」を参照してください。

(7)　代替ユーザ認証の MQOPEN 命令オプション

MQOPEN 命令を使用してオブジェクトをオープンするとき，キューマネジャはユーザがオブジェクトをオープンする権限を持つかどうかを確認します。権限がない場合，命令は失敗します。

しかし，サーバプログラムではキューマネジャに，サーバ自身の権限ではなく，サービスの提供先であるユーザの権限を確認させたいことがあります。これを実行するには，MQOO_ALTERNATE_USER_AUTHORITY オプションを MQOPEN 命令に使用し，代替ユーザ識別子を MQOD 構造体の AlternateUserId フィールドに指定してください。

通常，サーバは処理するメッセージにあるコンテキスト情報を基にユーザ識別子を取得します。

(8)　キューマネジャ停止状態の MQOPEN 命令オプション

TP1/Message Queue はキューマネジャの停止状態と MQOO_FAIL_IF_QUIESCING オプションをサポートしません。

8.3 動的キューの作成

アプリケーション終了後にキューが不要な場合は，動的キューを使用してください。例えば，応答キューとして動的キューを使用できます。応答キューの名前は，キューにメッセージを登録するときに，MQMD 構造体の ReplyToQ フィールドに指定します。

動的キューを作成するには，MQOPEN 命令でモデルキューというひな形を使用してください。モデルキューは，MQA サービス定義の mqaqueatl 定義コマンドで指定します。作成する動的キューはモデルキューの属性を取得します。

MQOPEN 命令の発行時に，MQOD 構造体の ObjectName フィールドにモデルキューの名前を指定してください。命令が完了すると，ObjectName フィールドには，作成された動的キューの名前が設定されます。また，ObjectQMgrName フィールドには，ローカルキューマネジャの名前が設定されます。

作成する動的キューの名前を指定する方法には，次に示す三つがあります。

- MQOD 構造体の DynamicQName フィールドに完全な名前を指定します。
- 名前のプリフィクス（32 文字以下）を指定し，キューマネジャに残りの名前を作成させます。キューマネジャがユニークな名前を生成しますが，特定のプリフィクスを使用して生成することもできます。この方法を使用するには，DynamicQName フィールドに指定する最後の文字（空白以外）にアスタリスクを指定してください。
- キューマネジャに完全な名前を作成させます。この方法を使用するには，DynamicQName フィールドの最初の文字位置にアスタリスクを指定します。

8.4　リモートキューのオープン

リモートキューは，アプリケーションの接続先以外のキューマネジャによって保持されるキューです。

リモートキューをオープンするには，ローカルキューの場合と同様に MQOPEN 命令を使用してください。キューの名前を指定する方法を次に示します。

- MQOD 構造体の ObjectName フィールドに，ローカルキューマネジャが認識するリモートキューの名前を指定します。

  注意事項

  この場合，ObjectQMgrName フィールドは空白のままにします。

- MQOD 構造体の ObjectName フィールドに，リモートキューマネジャが認識するリモートキューの名前を指定します。ObjectQMgrName フィールドには，次に示すどちらかを指定します。
  - リモートキューマネジャと同じ名前を持つ転送キューの名前。名前の大文字と小文字が正確に一致するよう注意してください。
  - あて先キューマネジャまたは転送キューに解決される，キューマネジャの別名。

  これによって，キューマネジャにメッセージのあて先，およびあて先に対応する転送キューを示すことになります。

- DefXmitQName がサポートされる場合は，MQOD 構造体の ObjectName フィールドにリモートキューマネジャが認識するリモートキュー名を指定してください。

  注意事項

  ObjectQMgrName フィールドに，リモートキューマネジャの名前を設定する必要があります。空白のままにしないでください。

MQOPEN 命令発行時にはローカル名だけが確認されます。使用する転送キューの有無は最後に確認されます。

8.5 MQCLOSE 命令によるオブジェクトのクローズ

オブジェクトをクローズするには，MQCLOSE 命令を使用してください。オブジェクトがキューの場合は，次に示す項目について注意してください。

- クローズする前に一時的動的キューを空にする必要はありません。
  一時的動的キューをクローズするとき，キューは格納されているメッセージとともに削除されます。コミット待ちの MQGET 命令，MQPUT 命令，または MQPUT1 命令がキューにある場合も同じです。
- MQOO_BROWSE オプションを使用してキューをオープンした場合，検索カーソルは削除されます。

クローズは同期点とは無関係です。同期点の前後でキューをクローズできます。

MQCLOSE 命令の入力として，次に示す項目を指定してください。

- コネクションハンドル
  オープンに使用したのと同じコネクションハンドルを使用してください。
- クローズしたいオブジェクトのハンドル
  MQOPEN 命令の出力から取得してください。
- Options フィールドの MQCO_NONE
  永続的動的キューをクローズする場合は除きます。
- 永続的動的キューをクローズする場合にメッセージがまだキューに残っていてもキューを削除するかどうかキューマネジャに指示する制御オプション

MQCLOSE 命令の出力を次に示します。

- 完了コード
- 理由コード
- オブジェクトハンドル
  MQHO_UNUSABLE_HOBJ にリセットされます。

9 キューへのメッセージ登録

この章では，キューへのメッセージ登録について説明します。

9.1　キューへのメッセージ登録の概要

メッセージをキューに登録するには，MQPUT 命令を使用してください。同じキューに複数のメッセージを登録する場合，MQOPEN 命令後に MQPUT 命令を繰り返し使用します。キューへのメッセージの登録が終わったら，MQCLOSE 命令を発行してください。

1 メッセージをキューに登録した直後にキューをクローズしたい場合は，MQPUT1 命令を使用できます。MQPUT1 命令は次に示す命令の発行順序と同等の動作をします。

1. MQOPEN 命令
2. MQPUT 命令
3. MQCLOSE 命令

一般的には，キューに登録したい複数のメッセージがある場合は，MQPUT 命令を使用する方が効率的です。どちらを使用するかはメッセージ長や環境によって異なります。

9.2 MQPUT 命令によるローカルキューへのメッセージ登録

MQPUT 命令の入力として，次に示す項目を指定してください。

- コネクションハンドル（HCONN）
- キューハンドル（Hobj）
- キューに登録したいメッセージのメッセージ記述子（MQMD 構造体）
- 制御情報
  メッセージ登録オプション（MQPMO 構造体）
- メッセージに含まれるデータ長（MQLONG）
- メッセージデータ本体

MQPUT 命令の出力を次に示します。

- 理由コード（MQLONG）
- 完了コード（MQLONG）

成功する場合，命令はメッセージ登録オプションとメッセージ記述子にも値を返します。命令はオプションを更新し，メッセージが送信されたキューマネジャとキューの名前が設定されます。登録したメッセージの識別子に対してユニークな値をキューマネジャに生成させたい場合は，2 進数の 0 を MQMD 構造体の MsgId フィールドに指定してください。ユーザに構造体を返す前に，命令は MsgId フィールドに値を設定します。この値は次の MQPUT 命令を発行する前にリセットしてください。

9.2.1 ハンドルの指定

TP1/Message Queue では，MQCONN 命令で返されたコネクションハンドルを使用してください。

環境に関係なく，MQOPEN 命令で返された同じキューハンドルを使用してください。

9.2.2 MQMD 構造体によるメッセージの定義

メッセージ記述子（MQMD 構造体）は，MQPUT 命令と MQPUT1 命令で使用する入出力用のパラメタです。キューに登録するメッセージを定義します。

メッセージに MQPRI_PRIORITY_AS_Q_DEF または MQPER_PERSISTENCE_AS_Q_DEF が指定され，キューがクラスタキューの場合には，MQPUT 命令が値を解決します。MQPUT 命令で該当するキューが無効な場合，命令は失敗します。

注意

MsgId および CorrelId フィールドをユニークにするには，新しいメッセージの登録前に MQPMO_NEW_MSG_ID または MQPMO_NEW_CORREL_ID を使用します。これらのフィールドの値は MQPUT 命令の成功時に返されます。

9.2.3 MQPMO 構造体によるオプションの指定

MQPMO 構造体は，MQPUT 命令と MQPUT1 命令にオプションを渡すために使用します。

次に示すフィールドがあります。

- StrucId
- Version
- Options
- Context
- ResolvedQName
- ResolvedQMgrName

各フィールドについて説明します。

StrucId

メッセージ登録オプション構造体を識別します。4 文字のフィールドです。常に MQPMO_STRUC_ID を指定してください。

Version

構造体のバージョン番号を記述します。デフォルトは MQPMO_VERSION_1 です。
MQPMO_VERSION_2 を指定すると，配布リストを使用できます。
MQPMO_CURRENT_VERSION を指定すると，アプリケーションは常に最新のバージョンを使用します。

Options

次に示す項目を制御します。

- 登録操作がトランザクションに参加するかどうか
- メッセージに関連づけるコンテキスト情報
- コンテキスト情報の取得先
- キューマネジャが停止状態のときに命令が失敗するかどうか
- グループ化とセグメント分割を許可するかどうか
- 新規のメッセージ識別子と相関識別子の生成
- メッセージとセグメントがキューに登録される順序

Options フィールドをデフォルトの値（MQPMO_NONE）にする場合は，関連するデフォルトのコンテキスト情報が登録するメッセージに設定されます。

Context

Options フィールドで要求した場合に，コンテキスト情報のコピー元とするキューハンドルの名前です。

ResolvedQName

メッセージを受信するため，別名の解決後にオープンされたキューの名前です。出力フィールドです。

ResolvedQMgrName

ResolvedQName フィールドのキューを別名の解決後に保持するキューマネジャの名前です。出力フィールドです。

MQPMO 構造体は配布リストで必要なフィールドも提供します。配布リストを使用する場合はバージョン 2 の MQPMO 構造体を使用してください。

バージョン 2 の MQPMO 構造体には，次に示すフィールドがあります。

Version

構造体のバージョン番号を記述します。配布リストでは，MQPMO_VERSION_2 を指定します。

RecsPresent

配布リストにあるキューの数です。つまり，登録メッセージレコード（MQPMR 構造体）および応答レコード（MQRR 構造体）の数です。

指定した値は，MQOPEN 命令で提供されたオブジェクトレコードの数と同じになることがあります。しかし，MQOPEN 命令で提供されたオブジェクトレコードの数よりも値が小さい場合，または登録メッセージレコードが提供されない場合には，メッセージ記述子によって提供されるデフォルトの値から，定義されないキューの値が取得されます。また，提供されたオブジェクトレコードの数よりも値が大きい場合には，余分な登録メッセージレコードは無視されます。

次に示すどちらかを実行することをお勧めします。

- 各あて先から報告または応答を受信したい場合は，MQOR 構造体と同じ値を入力して，MQPMR 構造体にある MsgId フィールドを使用してください。どちらの場合でも MsgId フィールドを 0 にするか，または MQPMO_NEW_MSG_ID を指定します。

  キューにメッセージを登録したときは，キューマネジャで生成した MsgId フィールドの値を MQPMR 構造体で利用できます。ユーザは各報告および応答に対応するあて先を識別するために，これらの値を使用できます。

- 報告または応答を受信しない場合は，次に示すどちらかを選択してください。

  1.失敗したあて先を即時に識別したい場合は，RecsPresent フィールドに MQOR 構造体と同じ値を入力して，MQRR 構造体を指定してください。MQPMR 構造体は指定しないでください。

  2.失敗したあて先を識別しない場合は，RecsPresent フィールドに 0 を入力してください。MQPMR 構造体および MQRR 構造体は指定しないでください。

注意

MQPUT1 命令を使用している場合は，応答レコードポインタおよび応答レコードオフセットの数は 0 にしてください。

PutMsgRecFields

各登録メッセージレコード（MQPMR 構造体）にあるフィールドを示します。

該当するフィールドのリストについては，「9.6.2(1)　MQPMR 構造体の使用」を参照してください。

PutMsgRecPtr および PutMsgRecOffsest

登録メッセージレコードの位置を示すために，ポインタ（C 言語）およびオフセット（COBOL 言語）が使用されます。

最初の登録メッセージレコードのポインタを指定するには PutMsgRecPtr フィールドを使用してください。最初の登録メッセージレコードのオフセットを指定するには PutMsgRecOffsest フィールドを使用してください。これは MQPMO 構造体の先頭からのオフセットです。PutMsgRecFields フィールドによっては，PutMsgRecPtr または PutMsgRecOffsest フィールドにヌル以外の値を入力してください。

ResponseRecPtr および ResponseRecOffset

応答レコードの位置を示すために，ポインタ（C 言語）およびオフセット（COBOL 言語）が使用されます。

最初の応答レコードのポインタを指定するには ResponseRecPtr フィールドを使用してください。最初の応答レコードのオフセットを指定するには ResponseRecOffset フィールドを使用してください。これは MQPMO 構造体の先頭からのオフセットです。ResponseRecPtr または ResponseRecOffset フィールドにはヌル以外の値を入力してください。

注意

配布リストにメッセージを登録するために MQPUT1 命令を使用している場合は，ResponseRecPtr フィールドはヌルまたは 0 にし，ResponseRecOffset フィールドは 0 にしてください。

配布リストにメッセージを登録する方法については，「9.6　配布リスト」を参照してください。

9.2.4　ユーザメッセージ内のデータ

ユーザデータのあるバッファのアドレスは，MQPUT 命令の Buffer パラメタに指定してください。メッセージ内には任意のデータを入れることができます。ただし，データ長は，メッセージを処理するアプリケーションの効率に影響します。

データの最大長は次に示す項目によって制限されます。

- キューマネジャの MaxMsgLength 属性
- メッセージを登録するキューの MaxMsgLength 属性
- システムが追加するメッセージヘッダの長さ

  デッドレターヘッダである MQDLH 構造体や配布リストヘッダである MQDH 構造体です。

キューマネジャの MaxMsgLength 属性は，キューマネジャが処理できるメッセージ長です。TP1/Message Queue のデフォルトは 4096000 バイトです。この属性を照会するにはキューマネジャオブジェクトに MQINQ 命令を使用してください。長大なメッセージに対してはこの値を変更できます。

キューの MaxMsgLength 属性は，キューに登録できるメッセージの最大長です。この属性の値よりも大きなメッセージを登録すると，MQPUT 命令は失敗します。リモートキューにメッセージを登録する場合には，リモートキュー，あて先までにある中間の転送キュー，および使用するチャネルの MaxMsgLength 属性によって成功と失敗が決まります。

MQPUT 命令では，メッセージ長はキューおよびキューマネジャの MaxMsgLength 属性値以下にしてください。両方の値は別物ですが，キューの MaxMsgLength 属性値をキューマネジャ以下にすることをお勧めします。

TP1/Message Queue では次に示す環境でヘッダ情報を追加します。

- リモートキューにメッセージを登録する場合

  TP1/Message Queue は MQXQH 構造体をメッセージに追加します。この構造体にはあて先キューとそれを保持するキューマネジャの名前が記述されます。

- リモートキューにメッセージを転送できない場合

  TP1/Message Queue はデッドレターキューにメッセージを登録します。メッセージには MQDLH 構造体が追加されます。この構造体にはあて先キューの名前とデッドレターキューに登録された理由が記述されます。

- 複数のあて先キューにメッセージを登録する場合

  TP1/Message Queue は MQDH 構造体をメッセージに追加します。この構造体には，転送キューにある配布リストに属するメッセージのデータが記述されます。最適なメッセージ長を選択するときには，この点を考慮してください。

- セグメント分割されたメッセージまたはグループ内のメッセージの場合

  システムは MQMDE 構造体を付加します。

メッセージ長がキューの最大長と同じ場合には，ヘッダが追加されるとメッセージが長大になって，登録操作は失敗します。登録操作失敗の可能性を減らすには，次に示す方法があります。

- 転送キューおよびデッドレターキューの MaxMsgLength 属性値よりも，メッセージ長を小さくしてください。少なくとも，MQ_MSG_HEADER_LENGTH の値分小さくしてください。長大な配布リストを使用する場合はさらに小さくしてください。
- デッドレターキューの MaxMsgLength 属性値をそれを保持するキューマネジャの値と同じにしてください。

オブジェクトの属性と定数値については，マニュアル「TP1/Message Queue プログラム作成リファレンス」を参照してください。

9.3　リモートキューへのメッセージ登録

ローカルキューではなく，自アプリケーションが接続しているキューマネジャとは異なるキューマネジャによって保持されるリモートキューにメッセージを登録する場合，オープンするキュー名の指定には特に注意してください。詳細については，「8.4　リモートキューのオープン」を参照してください。ローカルキューに MQPUT 命令または MQPUT1 命令を発行するのと変わりありません。

9.4　コンテキスト情報の制御

コンテキスト情報を制御するには，MQPMO 構造体の Options フィールドを使用してください。

Options フィールドを使用しない場合は，メッセージに設定されたメッセージ記述子中の識別情報とコンテキスト情報を，キューマネジャが上書きします。これは MQPMO_DEFAULT_CONTEXT オプションを指定する場合と同じです。照会画面のユーザ入力を処理するためなどに新規メッセージを作成するときに，このデフォルトコンテキスト情報を使用できます。

メッセージに関連するコンテキスト情報が不要な場合は，MQPMO_NO_CONTEXT オプションを使用してください。

9.4.1　識別コンテキストの渡し方

通常，データが最終あて先に到着するまでの間に，アプリケーション間でメッセージからメッセージへと識別コンテキスト情報が渡されます。アプリケーションはデータを変更するたびに登録元コンテキスト情報を変更します。しかし，コンテキスト情報を変更したり設定したりするアプリケーションは，適切な権限を持っていなければなりません。キューマネジャは，アプリケーションがキューをオープンするときに，この権限を確認します。アプリケーションは MQOPEN 命令を発行するために適切なコンテキストオプションを使用する権限を持っていなければなりません。

アプリケーションでメッセージを取得し，メッセージのデータを処理し，変更したデータを異なるメッセージに（他アプリケーションでの処理のためなどに）登録する場合，アプリケーションは識別コンテキスト情報を登録元メッセージから新規メッセージに渡さなければなりません。キューマネジャには登録元コンテキスト情報を生成させることができます。

登録元メッセージのコンテキスト情報を保存するには，メッセージ取り出し用にキューをオープンするときに MQOO_SAVE_ALL_CONTEXT オプションを使用してください。このオプションはほかのオプションに追加して MQOPEN 命令で使用できます。ただし，メッセージの検索だけの場合には，コンテキスト情報を保存できないことに注意してください。

2 番目のメッセージを作成するとき，次に示す項目を実行してください。

- MQOO_PASS_IDENTITY_CONTEXT オプションを使用してキューをオープンします。MQOO_OUTPUT オプションに追加します。
- MQPMO 構造体の Context フィールドに，コンテキスト情報を保存したキューのハンドルを指定します。
- MQPMO 構造体の Options フィールドに，MQPMO_PASS_IDENTITY_CONTEXT オプションを指定します。

9.4.2　すべてのコンテキストの渡し方

アプリケーションでメッセージを取り出して，ほかのメッセージにメッセージデータを変更しないで登録する場合には，アプリケーションは識別コンテキスト情報と登録元コンテキスト情報の両方を登録元メッセージから新規メッセージに渡さなければなりません。そのようなアプリケーションの例としては，メッセージ移動アプリケーションがあります。これはキュー間でメッセージを移動させるアプリケーションです。

識別コンテキストを渡すときと同じ処理をしてください。ただし，MQOPEN 命令には MQOO_PASS_ALL_CONTEXT オプションを指定し，メッセージ登録には MQPMO_PASS_ALL_CONTEXT オプションを指定します。

9.4.3　識別コンテキストの設定

メッセージに識別コンテキスト情報を設定する場合に，キューマネジャに登録元コンテキスト情報を設定させるには，次に示す項目を実行します。

- MQOO_SET_IDENTITY_CONTEXT オプションを使用してキューをオープンします。
- MQPMO_SET_IDENTITY_CONTEXT オプションを指定して，メッセージをキューに登録します。メッセージ記述子には，必要な識別コンテキスト情報を指定してください。

9.4.4　すべてのコンテキストの設定

識別コンテキスト情報と登録元コンテキスト情報の両方をメッセージに設定する場合は，次に示す項目を実行します。

- MQOO_SET_ALL_CONTEXT オプションを使用してキューをオープンします。
- MQPMO_SET_ALL_CONTEXT オプションを指定してキューにメッセージを登録します。メッセージ記述子には，必要な識別コンテキスト情報と登録元コンテキスト情報を指定します。

どちらのコンテキストの設定にも適切な権限が必要です。

9.5 MQPUT1 命令によるキューへの 1 メッセージの登録

1 メッセージを登録したあとでキューを即時にクローズするには，MQPUT1 命令を使用してください。例えば，サーバアプリケーションで各キューに応答メッセージを送信する場合などに MQPUT1 命令を使用します。

MQPUT1 命令は，MQOPEN 命令，MQPUT 命令，および MQCLOSE 命令の順で命令を発行するのと機能的に同等です。MQPUT 命令と MQPUT1 命令の違いは，MQPUT 命令ではオブジェクトハンドルの指定が必要で，MQPUT1 命令では MQOPEN 命令で定義するのと同じオブジェクト記述子の指定が必要ということです。オブジェクト記述子（MQOD 構造体）を指定することによって，オープンするキューについての情報を MQPUT1 命令に設定します。

MQPUT1 命令の入力として，次に示す項目を指定してください。

- コネクションハンドル（HCONN）
- オープンしたいキューのオブジェクト記述子（MQOD 構造体）
- キューに登録したいメッセージのメッセージ記述子（MQMD 構造体）
- 制御情報
  メッセージ登録オプション（MQPMO 構造体）
- メッセージに含まれるデータ長（MQLONG）
- メッセージデータ本体

MQPUT1 命令の出力を次に示します。

- 理由コード（MQLONG）
- 完了コード（MQLONG）

成功する場合，命令はメッセージ登録オプションとメッセージ記述子にも値を返します。命令はオプションを更新し，メッセージが送信されたキューマネジャとキューの名前が設定されます。登録したメッセージの識別子に対してユニークな値をキューマネジャに生成させたい場合は，2 進数の 0 を MQMD 構造体の MsgId フィールドに指定してください。ユーザに構造体を返す前に，命令は値を MsgId フィールドに設定します。

注意

MQPUT1 命令にモデルキュー名は使用できません。ただし，モデルキューをオープンしたあとは，MQPUT1 命令を動的キューに発行できます。

MQPUT1 命令の入力パラメタについて説明します。

Hconn

コネクションハンドルです。MQCONN 命令で返されたコネクションハンドルを指定してください。

ObjDesc

オブジェクト記述子（MQOD 構造体）です。

ObjectName および ObjectQMgrName フィールドには，メッセージを登録したいキューの名前とそれを保持するキューマネジャの名前を指定してください。

DynamicQName フィールドは，MQPUT1 命令ではモデルキューが使用できないので無視されます。

キューをオープンするための権限を確認するのに使用する代替ユーザ識別子を指定するときは，AlternateUserId フィールドを使用してください。

MsgDesc

メッセージ記述子（MQMD 構造体）です。MQPUT 命令と同様に，キューに登録するメッセージを定義するのに使用してください。

PutMsgOpts

メッセージ登録オプション（MQPMO 構造体）です。MQPUT 命令と同様に使用します。

Options フィールドが 0 に設定される場合，キューにアクセスする権限を確認するときに，キューマネジャはユーザ独自のユーザ ID を使用します。また，キューマネジャは MQOD 構造体の AlternateUserId フィールドにある代替ユーザ識別子を無視します。

BufferLength

ユーザメッセージの長さです。

Buffer

メッセージのテキストがあるバッファ領域です。

クラスタを使用するとき，MQPUT1 命令は MQOO_BIND_NOT_FIXED が有効であるのと同じように動作します。アプリケーションは MQOD 構造体ではなく，MQPMO 構造体の解決されたフィールドを使用して，メッセージの送信先を決定します。

9.6 配布リスト

配布リストを使用すると，1 回の MQPUT 命令または MQPUT1 命令で複数のあて先にメッセージを登録できます。1 回の MQOPEN 命令で複数のキューをオープンでき，その後，1 回の MQPUT 命令で各キューにメッセージを登録できます。その処理で使用される構造体の一般情報は，配布リストにある各あて先に関連する固有情報に置き換えられます。

MQOPEN 命令を発行するとき，一般情報がオブジェクト記述子（MQOD 構造体）から取得されます。Version フィールドに MQOD_VERSION_2 を指定し，RecsPresent フィールドに 0 より大きい値を指定する場合は，Hobj パラメタはキューハンドルではなく，一つ以上のキューについての配布リストのハンドルとして定義されます。このとき，固有情報がオブジェクトレコード（MQOR 構造体）から取得されます。MQOR 構造体にはあて先の詳細（ObjectName と ObjectQMgrName フィールド）があります。

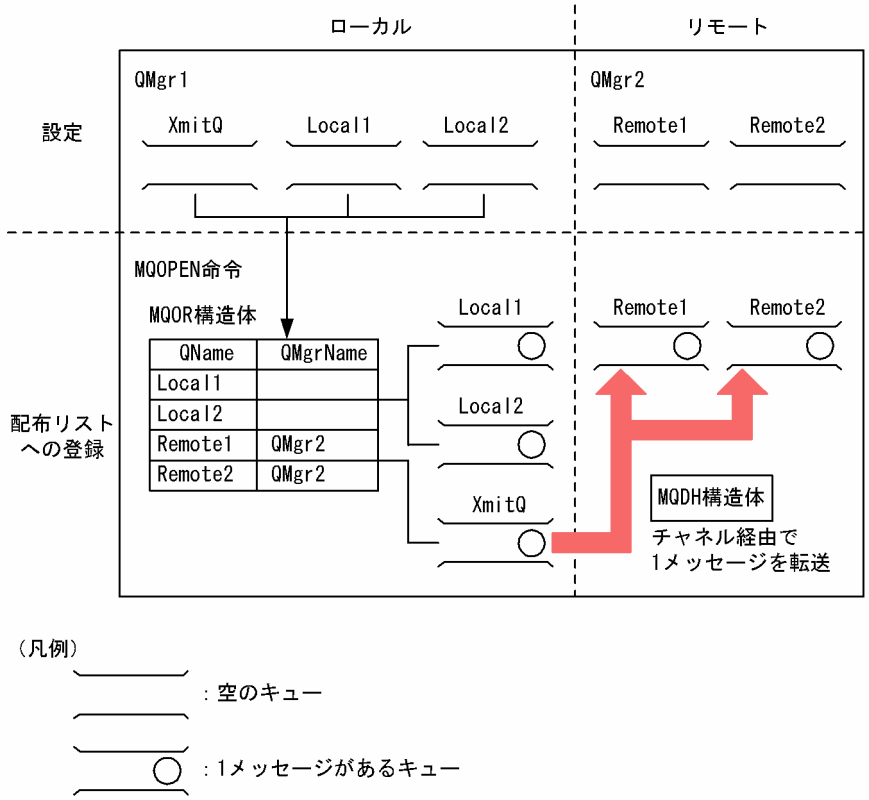
オブジェクトハンドル（Hobj）は MQPUT 命令に渡されます。これによってユーザは一つのキューにではなく，配布リストにメッセージを登録できます。

メッセージをキューに登録（MQPUT 命令）するとき，一般情報がメッセージ登録オプション（MQPMO 構造体）およびメッセージ記述子（MQMD 構造体）から取得されます。固有情報は登録メッセージレコード（MQPMR 構造体）から取得されます。

応答レコード（MQRR 構造体）では各あて先キューに固有な完了コードと理由コードを受け取れます。

配布リストの動作について，次の図に示します。チャネルを経由して 1 メッセージが転送され，複数のリモートキューに登録されています。

図 9-1　配布リストの動作



9.6.1　配布リストのオープン

配布リストをオープンするには MQOPEN 命令を使用し，配布リストで実行したいことを指定するには MQOPEN 命令のオプションを使用してください。

MQOPEN 命令の入力として，次に示す項目を指定してください。

- コネクションハンドル
- オブジェクト記述子（MQOD 構造体）の一般情報
- オープンする各キューの名前
  オブジェクトレコード（MQOR 構造体）を使用します。

MQOPEN 命令の出力を次に示します。

- 配布リストのアクセスを表すオブジェクトハンドル
- 一般完了コード
- 一般理由コード
- 応答レコード（任意）
  各あて先についての完了コードと理由コードがあります。

(1) MQOD 構造体の使用

オープンするキューを識別するには MQOD 構造体を使用してください。配布リストを定義するには，Version フィールドに MQOD_VERSION_2 を，RecsPresent フィールドに 0 より大きい値を，ObjectType フィールドに MQOT_Q を指定してください。

(2) MQOR 構造体の使用

MQOR 構造体を各あて先に提供してください。この構造体にはあて先キューとキューマネジャの名前があります。配布リストでは MQOD 構造体の ObjectName および ObjectQMgrName フィールドは使用しません。一つ以上のオブジェクトレコードが必要です。ObjectQMgrName フィールドが空白のままの場合はローカルキューマネジャが使用されます。

あて先キューは次に示す 2 とおりの方法で指定できます。

- ObjectRecPtr フィールドを使用する方法

  アプリケーションで MQOD 構造体とは別に MQOR 構造体の配列を宣言し，ObjectRecPtr フィールドに配列のアドレスを設定してください。C 言語での記述例を次に示します。

```
MQOD MyMqod;
MQOR MyMqor[100];
MyMqod.ObjectRecPtr = MyMqor;
```

  ObjectRecPtr フィールドは，ポインタデータ型をサポートするプログラミング言語（C 言語）で使用してください。

- ObjectRecOffset フィールドを使用する方法

  アプリケーションで MQOD 構造体に，必要な配列要素数の MQOR 構造体が続く独自の構造体を宣言してください。また，ObjectRecOffset フィールドには，配列の最初の要素について MQOD 構造体の最初からのオフセットを設定します。オフセットが正確であるように注意してください。

  アプリケーションのすべての動作環境で使用できる場合は，プログラミング言語で提供される組み込み機能を利用することをお勧めします。COBOL 言語での記述例を次に示します。

```
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.
```

  必要な組み込み機能を動作環境で使用できない場合には MQOD_CURRENT_LENGTH 定数を使用できます。この場合の記述例を次に示します。

```
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.
```

ただし，この方法は MQOD 構造体と MQOR 構造体の配列が連続しているときだけ，正常に動作します。コンパイラが MQOD 構造体と MQOR 構造体の配列の間にスキップバイトを挿入するときには，ObjectRecOffset フィールドの値にその分を追加してください。

ObjectRecOffset フィールドは，ポインタデータ型をサポートしていないプログラミング言語（COBOL 言語）で使用してください。

ObjectRecOffset または ObjectRecPtr フィールドのどちらかを使用してください。両方が 0 の場合，または両方が 0 でない場合には，MQRC_OBJECT_RECORDS_ERROR の理由コードで命令が失敗します。

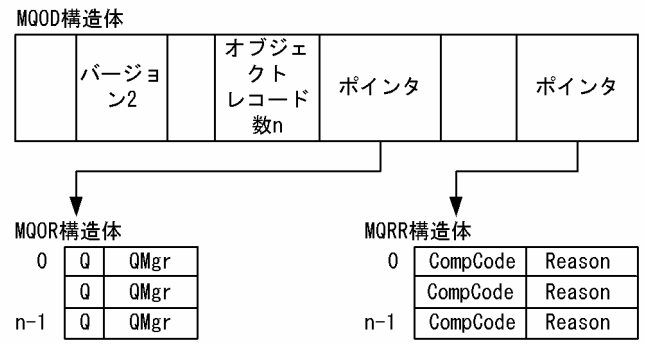
(3) MQRR 構造体の使用

この構造体は各あて先に固有な応答レコードです。応答レコードには，配布リストの各キューに対応する CompCode および Reason フィールドがあります。この構造体を使用してユーザは問題を特定できます。

例えば，MQRC_MULTIPLE_REASONS の理由コードを受け取り，配布リストに五つのあて先キューがある場合は，この構造体を使用しないとどのキューに問題があるのか特定できません。しかし，各あて先に完了コードと理由コードがあれば，エラーをより容易に特定できるようになります。

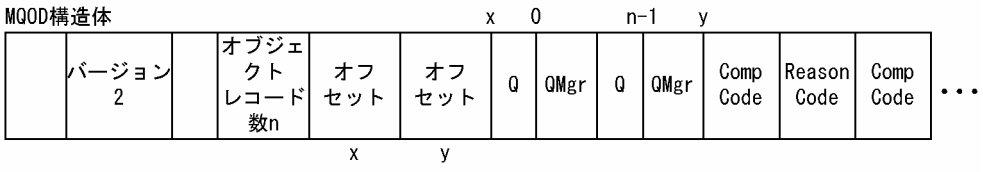
C 言語での配布リストのオープンについて，次の図に示します。

図 9-2 C 言語での配布リストのオープン



COBOL 言語での配布リストのオープンについて，次の図に示します。

図 9-3 COBOL 言語での配布リストのオープン



(4) MQOPEN 命令の使用

配布リストのオープン時には，次に示すオプションを指定します。

- MQOO_OUTPUT
- MQOO_FAIL_IF_QUIESCING（任意）
- MQOO_ALTERNATE_USER_AUTHORITY（任意）
- MQOO_*_CONTEXT（任意）

9.6.2　配布リストへのメッセージ登録

配布リストにメッセージを登録するには，MQPUT 命令または MQPUT1 命令を使用できます。入力として，次に示す項目を設定してください。

- コネクションハンドル
- オブジェクトハンドル
  配布リストを MQOPEN 命令でオープンした場合は，Hobj パラメタは配布リストへの登録だけを許可します。
- メッセージ記述子（MQMD 構造体）
- メッセージ登録オプション（MQPMO 構造体）の制御情報
- 登録メッセージレコード（MQPMR 構造体）の制御情報
- メッセージ中のデータ長（MQLONG）
- メッセージデータ本体

出力を次に示します。

- 完了コード
- 理由コード
- 応答レコード（オプション）

(1)　MQPMR 構造体の使用

この構造体は必要に応じて使用します。MQMD 構造体に定義済みの情報とは異なる情報を，あて先固有の情報としてフィールドに設定します。

各レコードの内容は，MQPMO 構造体の PutMsgRecFields フィールドによって決定されます。

指定例を次に示します。

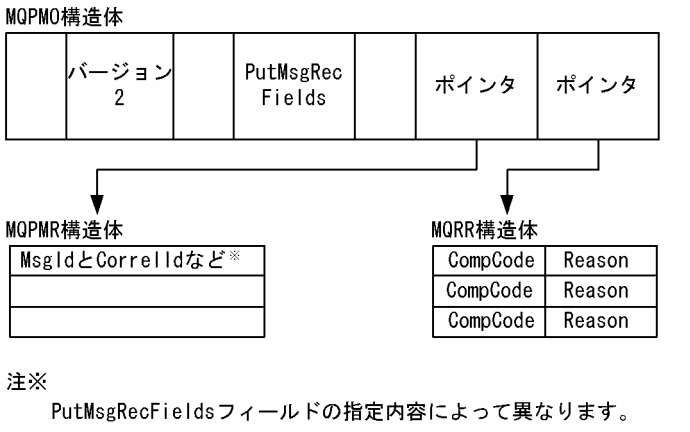
```
typedef struct
{
  MQBYTE24 MsgId;
  MQBYTE24 CorrelId;
} PutMsgRec;
/******************
```

```
PMO.PutMsgRecFields =
  MQPMRF_MSG_ID | MQPMRF_CORREL_ID;
```

ここでは，MsgId および CorrelId フィールドが配布リストの各あて先に設定されます。登録メッセージレコードは配列で指定されます。

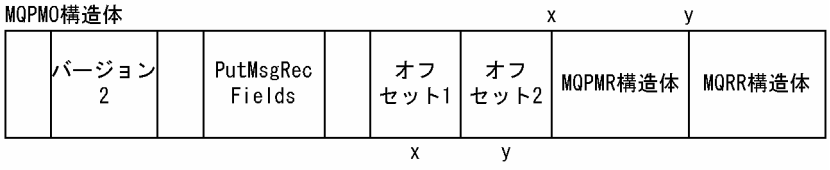
C 言語での配布リストへの登録について，次の図に示します。

図 9-4　C 言語での配布リストへの登録



COBOL 言語での配布リストへの登録について，次の図に示します。

図 9-5　COBOL 言語での配布リストへの登録



(2)　MQPUT1 命令の使用

MQPUT1 命令を使用する場合は，次に示す項目について注意してください。

1. ResponseRecOffset および ResponseRecPtr フィールドはヌルまたは 0 にしてください。
2. 応答レコードが必要な場合は，MQOD 構造体からの位置を指定してください。

9.7　登録に失敗する状況

メッセージを登録したいキューに登録が許可されていない場合には，MQPUT 命令と MQPUT1 命令は失敗し，MQRC_PUT_INHIBITED の理由コードを返します。他アプリケーションからキューの属性を定期的に変更するように設計しているときなどには，あとから実行することによってメッセージの登録を成功させることができます。

また，メッセージを登録したいキューが満杯の場合は，MQPUT 命令または MQPUT1 命令は失敗し，MQRC_Q_FULL を返します。

動的キュー（一時的動的キューまたは永続的動的キュー）が削除された場合，取得済みのオブジェクトハンドルを使用している MQPUT 命令は失敗し，MQRC_Q_DELETED の理由コードを返します。この場合，使用しなくなったオブジェクトハンドルをクローズすることをお勧めします。

配布リストを使用する場合には，一つの問い合わせで複数の完了コードと理由コードが発生することがあります。これらは MQOPEN 命令と MQPUT 命令の CompCode および Reason フィールドだけを使用しては対応できません。

複数のあて先にメッセージを登録するために配布リストを使用するときは，各あて先について，CompCode および Reason フィールドが応答レコードにあります。完了コードが MQCC_FAILED の場合は，あて先キューに何もメッセージが登録されていません。完了コードが MQCC_WARNING の場合は，一つ以上のあて先キューにメッセージの登録が成功しています。理由コードが MQRC_MULTIPLE_REASONS の場合は，各あて先についての理由コードが異なります。そこで，エラーが発生したキューを特定しその理由を知るために，MQRR 構造体を使用することをお勧めします。

10 キューからのメッセージ取り出し

この章では，キューからのメッセージ取り出しについて説明します。

10.1　キューからのメッセージ取り出しの概要

キューからメッセージを取り出すには二つの方法があります。

- 他アプリケーションがあとから参照できないように，キューからメッセージを削除する方法
- 元のメッセージをキューに残したまま，メッセージをコピーする方法。これを検索といいます。検索済みのメッセージは容易に削除できます。

両方とも MQGET 命令を使用しますが，最初にアプリケーションをキューマネジャに接続し，MQOPEN 命令を使用してキューをオープンしなければなりません。これらの操作については，「7.　キューマネジャの接続と切り離し」および「8.　オブジェクトのオープンとクローズ」を参照してください。

キューをオープンしたら，同じキューのメッセージを繰り返し検索または削除できます。キューから必要なメッセージをすべて取り出し終わったら，MQCLOSE 命令を発行してください。

10.2　MQGET 命令によるキューからのメッセージ取り出し

MQGET 命令は，オープンされたローカルキューからメッセージを取り出します。他システムにあるキューからはメッセージを取り出せません。

MQGET 命令の入力として，次に示す項目を指定してください。

- コネクションハンドル
- キューハンドル
- キューから取り出したいメッセージのメッセージ記述子（MQMD 構造体）
- 制御情報（MQGMO 構造体）
- メッセージを保持するために割り当てたバッファ長（MQLONG）
- メッセージを登録するバッファのアドレス

MQGET 命令の出力を次に示します。

- 理由コード（MQLONG）
- 完了コード（MQLONG）
- 命令成功時に指定バッファに格納されたメッセージ
- オプション構造体
  メッセージが取り出されたキューの名前が設定されます。
- メッセージ記述子
  取り出されたメッセージを記述するフィールドの定数が設定されます。
- メッセージ長（MQLONG）

10.2.1　取り出し時のコネクションハンドルの指定

TP1/Message Queue では，MQCONN 命令で返されたコネクションハンドルを使用してください。

環境に関係なく，MQOPEN 命令で返された同じキューハンドルを使用してください。

10.2.2　MQMD 構造体によるメッセージの定義と MQGET 命令

キューから取り出すメッセージを識別するには，メッセージ記述子（MQMD 構造体）を使用してください。MQMD 構造体は，MQGET 命令で使用する入出力用のパラメタです。

特定メッセージの取り出しを指定する方法については，「10.4　特定メッセージの取り出し」を参照してください。

特定メッセージを指定しない場合は，MQGET 命令は最初のメッセージを取り出します。メッセージ優先度，キューの MsgDeliverySequence 属性，および MQGMO_LOGICAL_ORDER オプションによるメッセージ順序の決定については，「10.3　メッセージがキューから取り出される順序」を参照してください。

注意

MQGET 命令を繰り返し使用したい場合（キューにあるメッセージを一とおり参照する場合など）は，各命令の発行後に MsgId および CorrelId フィールドにヌルを設定してください。これによって，取り出されたメッセージの識別子フィールドが消去されます。

メッセージをグループ化したい場合は，GroupId フィールドは同じグループ内で同じでなければなりません。これは，グループをまとめるために命令が前回と同じ識別子のメッセージを検索するためです。

10.2.3　MQGMO 構造体によるオプションの指定

MQGMO 構造体は，MQGET 命令にオプションを渡すための入出力用の値です。

次に示すフィールドがあります。

StrucId

メッセージ取り出しオプション構造体を識別します。4 文字のフィールドです。常に MQGMO_STRUC_ID を指定してください。

Version

構造体のバージョン番号を記述します。デフォルトは MQGMO_VERSION_1 です。バージョン 2 のフィールドを使用したい場合，または論理的順序でメッセージを取り出したい場合は，MQGMO_VERSION_2 を指定してください。

MQGMO_CURRENT_VERSION を指定する場合，アプリケーションは常に最新のバージョンを使用します。

Options

アプリケーションコード内で，必要なオプションを任意の順序で指定します。

Options フィールドは，次に示す項目を制御します。

- 完了前に MQGET 命令がメッセージのキューへの到着を待ち合わすか
- 登録操作がトランザクションに参加するか
- ファーストメッセージ機能を有効にし，非永続メッセージを同期点外で取り出すか
- キューからメッセージを削除するか，検索するだけか
- 検索カーソルまたはその他の基準によってメッセージを選択するか
- メッセージがバッファよりも長い場合に命令を成功させるか
- キューマネジャが停止状態のときに命令が失敗するか
- メッセージデータの変換をするか
- メッセージとセグメントをキューから取り出す順序

- 完全な論理メッセージだけを取り出すか
- グループ内の全メッセージが利用できるときだけに，グループ内のメッセージを取り出すか
- 論理メッセージ内の全セグメントが利用できるときだけに，論理メッセージ内のセグメントを取り出すか

Options フィールドをデフォルトの値に（MQGMO_NO_WAIT）する場合は，MQGET 命令は次に示すとおりに動作します。

- 選択基準に合うメッセージがキューにない場合，命令はメッセージの到着を待たないで即時に完了します。
- TP1/Message Queue では，同期点を指定した命令は無効になります。
- 選択されたメッセージはキューから検索ではなく削除されます。
- アプリケーションデータの変換は実行されません。
- メッセージがバッファよりも長い場合は，命令は失敗します。

WaitInterval

WaitInterval フィールドは，MQGMO_WAIT 指定時にメッセージのキューへの到着を MQGET 命令が待ち合わせる最大時間（ミリ秒）を指定します。時間内にメッセージが到着しない場合に命令は完了し，選択基準に合うメッセージがキューにないことを示す理由コードを返します。

Signal1

TP1/Message Queue ではサポートしません。

Signal2

TP1/Message Queue ではサポートしません。

ResolvedQName

ResolvedQName フィールドは，出力用のフィールドです。キューマネジャがメッセージを取り出したキューの名前を別名の解決後に返します。

MatchOptions

MatchOptions フィールドは，MQGET 命令の選択基準を制御します。

GroupStatus

GroupStatus フィールドは，取り出したメッセージがグループ内にあるかを示します。

SegmentStatus

SegmentStatus フィールドは，取り出した項目が論理メッセージのセグメントであるかを示します。

Segmentation

Segmentation フィールドは，取り出したメッセージにセグメント分割を許可するかを示します。

10.2.4 バッファ領域長の指定

MQGET 命令の BufferLength パラメタには，取り出したメッセージを保持するのに使用するバッファ領域長を指定してください。この領域長を決定する方法は三つあります。

1. アプリケーションで取り出すメッセージの長さが既知の場合，該当する長さを指定してください。

   メッセージがバッファ領域長よりも長い場合に MQGET 命令を完了させたいときには，MQGMO_ACCEPT_TRUNCATED_MSG を MQGMO 構造体に指定できます。このとき，次に示すとおりに動作します。

   - 可能な範囲でバッファにメッセージが格納されます。
   - 命令は警告の完了コードを返します。
   - メッセージはキューから削除（バッファ領域に格納できない部分は破棄）され，検索カーソルが検索時に前進します。
   - メッセージの実際の長さは DataLength パラメタに格納されます。

   MQGMO_ACCEPT_TRUNCATED_MSG の指定なしでも，警告付きで命令は完了します。しかし，メッセージはキューから削除されないで，検索カーソルは前進しません。

2. バッファの長さを見積もります。領域長には 0 バイトを指定することもあります。MQGMO_ACCEPT_TRUNCATED_MSG は指定しないでください。MQGET 命令が失敗（バッファが小さ過ぎるなど）する場合は，メッセージ長が命令の DataLength パラメタに返されます。バッファには可能な範囲のメッセージが格納されますが，命令の処理は完了しません。該当するメッセージの MsgId フィールドを保存してから，後続の MQGET 命令を発行してください。このときに，正しい長さのバッファ領域を指定し，最初の命令で取得した MsgId フィールドを指定します。

   他アプリケーションが同じキューを処理する場合には，自アプリケーションで後続の MQGET 命令を発行する前に，他アプリケーションが必要なメッセージを削除することもあります。自アプリケーションには，存在しないメッセージを検索するためのむだな時間が掛かります。これを回避するには，最初は BufferLength パラメタに 0 を指定して MQGMO_ACCEPT_TRUNCATED_MSG を使用して，必要なメッセージが見つかるまでキューを検索します。その後，MQGET 命令に MQGMO_MSG_UNDER_CURSOR を指定して再度発行してください。検索と取り出しの間に他アプリケーションがメッセージを削除する場合には，2 回目の MQGET 命令はキュー全体を検索しないで即時に失敗します。これはカーソル下にメッセージがないためです。

3. キューとキューマネジャの MaxMsgLength 属性は，受け付けることができるメッセージの最大長を定義します。メッセージ長がわからない場合は，MQINQ 命令を使用して MaxMsgLength 属性を照会し，その値をメッセージバッファの長さにしてください。

   性能を低下させないように，バッファの長さはできるだけ実際のメッセージ長に近づけてください。

注意

MQGET 命令および MQINQ 命令では，アプリケーションで動的または静的に確保した領域に，情報を返却する場合があります。

そのため，実際に確保した領域の大きさが MQGET 命令の BufferLength パラメタ，MQINQ 命令の IntAttrCount パラメタ，および MQINQ 命令の CharAttrs パラメタに指定された値より小さい場合，領域破壊など予期しない事象が発生するので動作は保証されません。

10.3　メッセージがキューから取り出される順序

ユーザは，メッセージをキューから取り出す順序を制御できます。

10.3.1　優先度

メッセージをキューに登録するとき，アプリケーションは優先度を割り当てます。同じ優先度のメッセージはコミット順ではなく到着順でキューに登録されます。

キューマネジャは厳密な FIFO 順，または優先度付きの FIFO 順でキューを保持します。これはキューの MsgDeliverySequence 属性で設定します。メッセージがキューに到着すると，同じ優先度の最終メッセージのあとに挿入されます。

アプリケーションは優先度を無視して，キューから最初のメッセージを取り出したり，特定メッセージを取り出したりできます。例えば，早期に送信した特定メッセージの応答を処理できます。詳細については，「10.4　特定メッセージの取り出し」を参照してください。

アプリケーションで一連のメッセージをキューに登録する場合，他アプリケーションから登録順と同じ順序で取り出せます。この場合，次に示す条件があります。

- メッセージはすべて同じ優先度であること
- メッセージはすべて同じトランザクション内で登録されたか，トランザクション外で登録されていること
- 登録元アプリケーションにとってのローカルキューであること

これらの条件が満たされないで，アプリケーションの取り出すメッセージに順序性が必要な場合には，メッセージデータ内に通番情報を取り込んだり，次メッセージの送信前に受信確認を送信したりするようにアプリケーションを作成してください。

10.3.2　論理的順序と物理的順序

メッセージは各優先度内で，論理的順序または物理的順序で格納されます。

- 物理的順序

  メッセージがキューに到着する順序です。
- 論理的順序

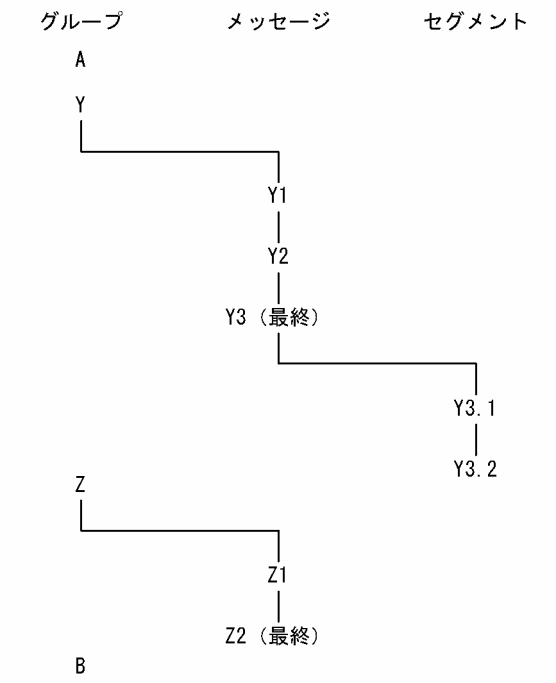
  グループ内のメッセージとセグメントのすべてに論理的な順序があり，互いに隣接するときの順序です。グループに属する最初の項目の物理的な位置によって決定されます。

グループ，メッセージ，およびセグメントの概要については，「3.6　メッセージグループ」を参照してください。物理的順序と論理的順序では次に示す点が異なります。

- グループは同じあて先に同時に異なるアプリケーションから到着することがあります。そのため，厳密な物理的順序は失われることがあります。
- 単一のグループ内のメッセージであっても，経路を設定し直されたり遅延したりすることによって，順序が前後することがあります。

キューでの論理的順序について，次の図に示します。

図 10-1　キューでの論理的順序

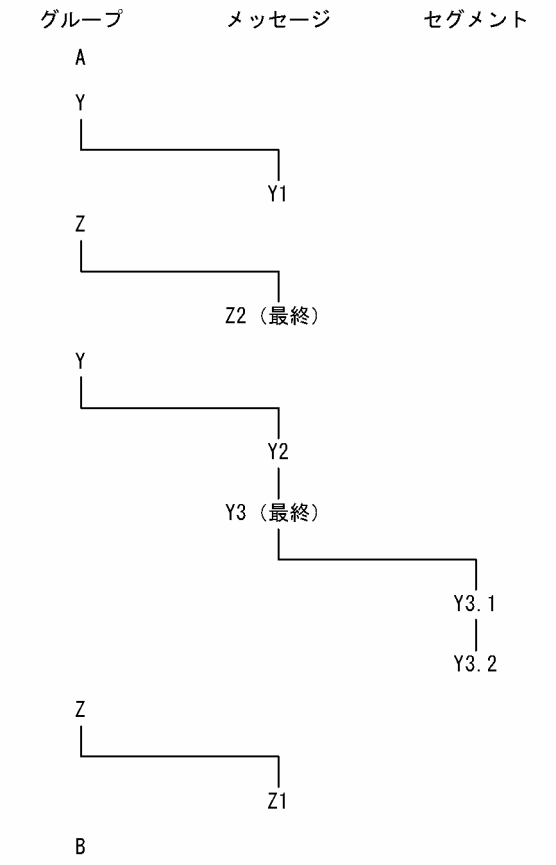


キューでのこれらのメッセージの順序を次に示します。

1. メッセージ A（グループにない）
2. グループ Y の論理メッセージ 1
3. グループ Y の論理メッセージ 2
4. グループ Y の（最終）論理メッセージ 3 のセグメント 1
5. グループ Y の（最終）論理メッセージ 3 の（最終）セグメント 2
6. グループ Z の論理メッセージ 1
7. グループ Z の（最終）論理メッセージ 2
8. メッセージ B（グループにない）

物理的順序は論理的順序とは異なります。各グループ内で最初の項目の物理的な位置が，グループ全体の論理的な位置を決定します。ここでは，グループ Y と Z が同時に到着し，グループ Z のメッセージ 2 が，同じグループのメッセージ 1 に先行する場合でのキューでの物理的順序について，次の図に示します。

図 10-2　キューでの物理的順序



キューでのこれらのメッセージの順序を次に示します。

1. メッセージ A（グループにない）
2. グループ Y の論理メッセージ 1
3. グループ Z の論理メッセージ 2
4. グループ Y の論理メッセージ 2
5. グループ Y の（最終）論理メッセージ 3 のセグメント 1
6. グループ Y の（最終）論理メッセージ 3 の（最終）セグメント 2
7. グループ Z の論理メッセージ 1
8. メッセージ B（グループにない）

メッセージを取り出すときに，物理的順序ではなく論理的順序でメッセージを取り出すことを指定するには，MQGMO_LOGICAL_ORDER を指定します。

MQGMO_BROWSE_FIRST および MQGMO_LOGICAL_ORDER を指定する MQGET 命令を発行する場合は，以降の MQGMO_BROWSE_NEXT 指定の MQGET 命令でも MQGMO_LOGICAL_ORDER を指定してください。逆に，MQGMO_LOGICAL_ORDER を指定しない MQGMO_BROWSE_FIRST の MQGET 命令を発行する場合は，以降の

MQGMO_BROWSE_NEXT 指定の MQGET 命令でも MQGMO_LOGICAL_ORDER を指定しないでください。

キューのメッセージを検索する MQGET 命令でキューマネジャが保存するグループ情報とセグメント情報は，キューからメッセージを削除する MQGET 命令でキューマネジャが保存する情報とは異なります。MQGMO_BROWSE_FIRST 指定時には，キューマネジャは検索用のグループ情報とセグメント情報を無視し，現在のグループと現在の論理メッセージがないかのようにキューを検索します。

注意

MQGMO_LOGICAL_ORDER を指定しないときにメッセージグループまたはグループ内にない論理メッセージの最後を越えて，MQGET 命令を使用して検索する場合には，特に注意が必要です。例えば，グループ内で最後のメッセージがグループ内で最初のメッセージよりキューで先行する場合に，グループの最後を越える MQGMO_BROWSE_NEXT を指定し，MsgSeqNumber フィールドに 1 を設定しながら MQMO_MATCH_MSG_SEQ_NUMBER を指定するときは，検索済みのグループ内で最初のメッセージが再度返されます。このことは，間に他グループを挟んで複数の MQGET 命令を発行する場合にも発生します。

キューを検索用に 2 回オープンすることによって，無限ループを回避できます。

- 各グループの最初のメッセージだけを検索するには，最初のハンドルを使用してください。
- 特定のグループ内のメッセージだけを検索するには，2 回目のハンドルを使用してください。
- 2 回目の検索カーソルを最初の検索カーソルの位置に移動するには，グループ内のメッセージを検索する前に，MQMO_*オプションを使用してください。
- グループの最後を越えて MQGMO_BROWSE_NEXT を使用しないでください。

通常は，検索時にアプリケーションで論理的順序または物理的順序を選択します。しかし，二つのモードを切り替える場合は，最初に検索を MQGMO_LOGICAL_ORDER で発行したときに論理的順序での位置が設定されることに注意してください。

検索カーソルがグループ内にいったん入ると，最初のメッセージが削除されても同じグループ内にあり続けます。また，最初の項目がないグループには MQGMO_LOGICAL_ORDER を使用して移動できません。

(1) 論理メッセージのグループ化

グループ内の論理メッセージを使用するのは，次に示す場合です。

- メッセージを正しい順序で処理したい場合
- グループ内の各メッセージを関連する方法で処理したい場合

どちらの場合も，グループ全体の取り出しは，同じ取り出しアプリケーションのインスタンスで実行してください。

例えば，グループを四つの論理メッセージで構成するとします。登録アプリケーションの記述例を次に示します。

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

dc_trn_begin()

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

dc_trn_unchained_commit()
```

取り出しアプリケーションではグループ内の全メッセージが到着するまで処理を開始しないことにします。そのため，グループ内の最初のメッセージには MQGMO_ALL_MSGS_AVAILABLE を指定します。グループ内の以降のメッセージではこのオプションは無視されます。

グループで最初の論理メッセージを取り出したら，グループの残りの論理メッセージを順に取り出すために MQGMO_LOGICAL_ORDER を使用します。

例えば，取り出しアプリケーションの記述例を次に示します。

```
/* グループ内の最初のメッセージ,
   またはグループ外のメッセージ待ち合わせ */
GMO.Options = MQGMO_SYNCPOINT | MQGMO_WAIT
            | MQGMO_ALL_MSGS_AVAILABLE | MQGMO_LOGICAL_ORDER
dc_trn_begin()
MQGET /* 先頭メッセージの処理 */
while ( GMO.GroupStatus == MQGS_MSG_IN_GROUP ) {
  MQGET
  /* グループ内の残りのメッセージの処理 */
  ...
}
if(GMO.GroupStatus != MQGS_LAST_MSG_IN_GROUP)
  MQGET /* MQGS_LAST_MSG_IN_GROUPメッセージの取り出し */

dc_trn_unchained_commit()
```

(2) 複数トランザクションにわたるグループの登録と取り出し

説明したように，すべてのグループが登録されトランザクションがコミットされるまでは，ノード（あて先がリモートの場合）を離れられないし，メッセージまたはセグメントの取り出しを開始できません。グループ全体を登録するのに時間が掛かる場合や，ノード上でキューのスペースが限られている場合には，不便なときがあります。これを回避するために，グループを複数のトランザクションで登録できます。

グループが複数のトランザクションで登録される場合には，登録アプリケーションの失敗が発生するときでも幾つかのグループでコミットが可能です。そのためにアプリケーションでは，ステータス情報を保存して各トランザクションをコミットすることによって，完了しなかったグループをあとで開始するときにステータス情報を使用できるようにします。最も簡単な方法はステータスキューにステータス情報を保存することです。グループ全体が登録に成功すると，ステータスキューは空になります。

セグメント分割を使用する場合も同様です。ステータス情報に Offset を含めてください。

複数のトランザクションに分けたグループの登録例を次に示します。

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

/* 最初のトランザクション */

dc_trn_begin()
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
StatusInfo = MQMD構造体のGroupIdとMsgSeqNumber
MQPUT（ステータス情報をステータスキューへ）PMO.Options =
                                              MQPMO_SYNCPOINT
dc_trn_chained_commit()

/* 2回目以降のトランザクション */

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQGET（ステータスキューから）GMO.Options = MQGMO_SYNCPOINT
StatusInfo = MQMD構造体のGroupIdとMsgSeqNumber
MQPUT（ステータス情報をステータスキューへ）PMO.Options =
                                              MQPMO_SYNCPOINT
dc_trn_chained_commit()

/* 最後のトランザクション */

MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
MQPUT MD.MsgFlags = MQMF_LAST_MSG_IN_GROUP
MQGET（ステータスキューから）GMO.Options = MQGMO_SYNCPOINT
dc_trn_unchained_commit()
```

すべてのトランザクションがコミットされると，グループ全体が成功し，ステータスキューは空になります。そうでない場合は，ステータス情報によって示される時点の状態で再度開始してください。
MQPMO_LOGICAL_ORDER は最初の登録時には使用できませんが，以降の登録で使用できます。

再度開始するときの記述例を次に示します。

```
dc_trn_begin()
MQGET（ステータスキューから）GMO.Options = MQGMO_SYNCPOINT
if (Reason == MQRC_NO_MSG_AVAILABLE)
  /* 通常の処理 */
  ...
else
  /* グループが早期に中断 */
  MQMD構造体のGroupIdとMsgSeqNumberにステータス情報の値を設定
  PMO.Options = MQPMO_SYNCPOINT
  MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP

  /* 通常処理を再度開始
     最終メッセージではない */
  PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT
  MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
```

```
  MQPUT MD.MsgFlags = MQMF_MSG_IN_GROUP
  StatusInfo = MQMD構造体のGroupIdとMsgSeqNumber
  MQPUT（ステータス情報をステータスキューへ）PMO.Options =
                                              MQPMO_SYNCPOINT
  dc_trn_unchained_commit()
```

取り出しアプリケーションでは，グループ全体が到着する前にメッセージの処理を開始できます。これによって，グループ内のメッセージについて応答時間を改善できます。また，グループ全体を格納する領域も不要になります。

回復のために，各メッセージはトランザクション内で取り出してください。また，上記の利点を使用できるように，メッセージの各グループに複数のトランザクションを使用してください。

対応する登録アプリケーションと同様に取り出しアプリケーションでも，各トランザクションがコミットされるときに自動的に保存されるステータス情報が必要です。最も簡単な方法はステータスキューに保存することです。グループ全体の処理に成功すると，ステータスキューは空になります。

注意

中間のトランザクションでは，各トランザクションで完全な新規メッセージを登録する代わりに，ステータスキューへの各 MQPUT 命令にメッセージのセグメントであることを指定（MQMF_SEGMENT を設定）することによって，ステータスキューからの MQGET 命令を回避できます。最後のトランザクションでは，最終セグメントをステータスキューに MQMF_LAST_SEGMENT 指定で登録し，その後の MQGMO_COMPLETE_MSG 指定の MQGET 命令でステータス情報を消去します。

再度開始するときの処理では，使用可能なステータスメッセージを取り出すため MQGET 命令を 1 回発行する代わりに，最終セグメントに届くまでステータスキューを MQGMO_LOGICAL_ORDER で検索してください。再度開始したあとの最初のトランザクションでは，ステータスセグメントを登録するときにオフセットを明示的に指定してください。

次に示す例では，1 グループのメッセージだけを考慮します。アプリケーションのバッファは，メッセージのセグメント分割の有無に関係なく，メッセージ全体を格納するのに十分な大きさがあるとします。そのため，各 MQGET 命令には MQGMO_COMPLETE_MSG を指定します。セグメント分割する場合にも同じ方法を適用できます。その場合は，ステータス情報に Offset を含めます。

単純化のため，一つのトランザクション内で最大四つのメッセージが取り出されるとします。

```
msgs = 0 /* トランザクション内での取り出しメッセージをカウント */
/* この時点でステータスメッセージなし */
dc_trn_begin()
MQGET
/* グループ内の残りのメッセージを取り出し */
while ( GroupStatus == MQGS_MSG_IN_GROUP ) {

  /* グループ内で最大4メッセージを処理 */
  GMO.Options = MQGMO_SYNCPOINT | MQGMO_WAIT
              | MQGMO_LOGICAL_ORDER
  while ( (GroupStatus == MQGS_MSG_IN_GROUP) && (msgs < 4) ) {
    MQGET
    msgs = msgs + 1
    /* 該当メッセージの処理 */
```

```
    ...
  /* whileの終わり */
  }

  /* 最終メッセージまたは4メッセージの取り出し完了 */
  /* グループの最終でない場合にステータスメッセージを更新 */
  MQGET（ステータスキューから）GMO.Options = MQGMO_SYNCPOINT
  if ( GroupStatus == MQGS_MSG_IN_GROUP )
    StatusInfo = MQMD構造体のGroupIdとMsgSeqNumber
    MQPUT（ステータス情報をステータスキューへ）PMO.Options =
                                                MQPMO_SYNCPOINT
  dc_trn_chained_commit()
  msgs = 0
/* whileの終わり */
}

If ( msgs > 0 )
  /* グループ内で1メッセージの場合はここ */
  dc_trn_chained_commit()
```

すべてのトランザクションがコミットされると，グループ全体が成功し，ステータスキューは空になります。コミットされない場合は，ステータス情報によって示される時点の状態で再度開始してください。MQGMO_LOGICAL_ORDER は最初の取り出し時には使用できませんが，以降の取り出しで使用できます。

再度開始するときの記述例を次に示します。

```
dc_trn_begin()
MQGET（ステータスキューから）GMO.Options = MQGMO_SYNCPOINT
if (Reason == MQRC_NO_MSG_AVAILABLE)
  /* 通常の処理 */
  ...
else
  /* グループが早期に中断 */
  /* グループの次メッセージは，ステータス情報からの
     シーケンス番号とグループIDに合わせて取り出して
     ください。 */
  GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT | MQGMO_WAIT
  MQGET GMO.MatchOptions = MQMO_MATCH_GROUP_ID |
        MQMO_MATCH_MSG_SEQ_NUMBER,
        MQMD.GroupId = ステータス情報からの値
        MQMD.MsgSeqNumber = ステータス情報からの値 + 1
  msgs = 1
  /* 該当メッセージの処理 */
  ...

  /* 通常処理を再度開始 */
  /* グループ内の残りのメッセージを取り出し */
  while ( GroupStatus == MQGS_MSG_IN_GROUP ) {

    /* グループ内で最大4メッセージを処理 */
    GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT |
                  MQGMO_WAIT | MQGMO_LOGICAL_ORDER
    while ( (GroupStatus == MQGS_MSG_IN_GROUP) && (msgs < 4) ) {
      MQGET
```

```
    msgs = msgs + 1
    /* 該当メッセージの処理 */
    ...

  /* 最終メッセージまたは4メッセージの取り出し完了 */
  /* グループの最終でない場合にステータスメッセージを更新 */
  MQGET（ステータスキューから）GMO.Options = MQGMO_SYNCPOINT
  if ( GroupStatus == MQGS_MSG_IN_GROUP )
    StatusInfo = MQMD構造体のGroupIdとMsgSeqNumber
    MQPUT（ステータス情報をステータスキューへ）PMO.Options =
                                              MQPMO_SYNCPOINT
  dc_trn_unchained_commit()
  msgs = 0
```

10.4　特定メッセージの取り出し

キューから特定メッセージを取り出すには，MQMD 構造体の MsgId および CorrelId フィールドを使用してください。しかし，アプリケーションで明示的にこれらのフィールドを設定できるので，ユーザが指定した値でユニークなメッセージを識別できないこともあります。MsgId および CorrelId フィールドを指定する取り出しについて，次の表に示します。MQGET 命令の GetMsgOpts パラメタに MQGMO_MSG_UNDER_CURSOR を指定する場合，これらのフィールドは無視されます。

表 10-1　MsgId および CorrelId フィールドを指定する取り出し

取り出されるメッセージ	MsgId	CorrelId
キューの最初のメッセージ	MQMI_NONE	MQCI_NONE
MsgId に合う最初のメッセージ	0 以外	MQCI_NONE
CorrelId に合う最初のメッセージ	MQMI_NONE	0 以外
MsgId と CorrelId の両方に合う最初のメッセージ	0 以外	0 以外

それぞれ，選択基準を満たす最初のメッセージが取り出されます。MQGMO_BROWSE_NEXT を指定する場合は，選択基準を満たす次のメッセージが取り出されます。

戻り時に MQGET 命令はメッセージに関連する MsgId と CorrelId フィールドを設定します。

MQMD 構造体の Version フィールドに 2 を設定する場合は，GroupId，MsgSeqNumber，および Offset フィールドを使用できます。一致オプションと取り出されるメッセージについて，次の表に示します。

表 10-2　一致オプションと取り出されるメッセージ

取り出されるメッセージ	一致オプション
キューの最初のメッセージ	MQMO_NONE
MsgId に合う最初のメッセージ	MQMO_MATCH_MSG_ID
CorrelId に合う最初のメッセージ	MQMO_MATCH_CORREL_ID
GroupId に合う最初のメッセージ	MQMO_MATCH_GROUP_ID
MsgSeqNumber に合う最初のメッセージ	MQMO_MATCH_MSG_SEQ_NUMBER
Offset に合う最初のメッセージ	MQMO_MATCH_OFFSET

注 1

MQMO_MATCH_XXX は，MQMD 構造体の XXX というフィールドに合うメッセージを指定します。

注 2

MQMO フラグは組み合わせて使用できます。例えば，MQMO_MATCH_GROUP_ID，MQMO_MATCH_MSG_SEQ_NUMBER，および MQMO_MATCH_OFFSET を同時に使用して，GroupId，MsgSeqNumber，および Offset フィールドに合うセグメントを指定できます。

注 3

MQGMO_LOGICAL_ORDER を指定する場合は，取り出しメッセージに影響があります。キューハンドル用に制御される情報にオプションが依存するからです。詳細については，「10.3.2　論理的順序と物理的順序」を参照してください。

注意

1. MQGET 命令は常に最初のメッセージをキューから取り出します。MQGET 命令を使用するときに特定メッセージを指定する場合，キューマネジャは該当メッセージを見つけるまでキューを検索します。そのため，アプリケーションの性能に影響することがあります。
2. MQMD 構造体のバージョン 2 を使用する場合は，MQMO_MATCH_MSG_ID および MQMO_MATCH_CORREL_ID を使用できます。このとき，MQGET 命令間で MsgId および CorrelId フィールドをリセットする必要はありません。

10.5　長大メッセージの処理

メッセージが，アプリケーション，キュー，またはキューマネジャにとって長過ぎることがあります。このような長大メッセージを処理するために，TP1/Message Queue は次に示す方法を提供します。

- キューとキューマネジャの MaxMsgLength 属性を増やす方法
- メッセージのセグメント分割をする方法
  メッセージはアプリケーションまたはキューマネジャによってセグメント分割されます。
- 参照メッセージを使用する方法

10.5.1　最大メッセージ長の増加

キューマネジャの MaxMsgLength 属性は，キューマネジャが処理できるメッセージの最大長を定義します。キューの MaxMsgLength 属性は，キューが処理できるメッセージの最大長を定義します。

TP1/Message Queue では，キューマネジャの MaxMsgLength 属性は mqaquemgr 定義コマンドの-l オプションで指定します。キューの MaxMsgLength 属性は mqaqueatl 定義コマンドの-l オプションで指定します。デフォルトの値は両方とも 4096000 バイトです。

変更するときには，メッセージ長が両方の MaxMsgLength 属性値以下になるようにしてください。しかし，既存のメッセージはどちらかの MaxMsgLength 属性値よりも長くなることがあります。定義済みキュー（mqamkque コマンドで作成したキュー）の属性変更は，mqachgque コマンドで実行してください。

メッセージがキューに長過ぎる場合には，MQRC_MSG_TOO_BIG_FOR_Q が返されます。同様にメッセージがキューマネジャに長過ぎる場合には，MQRC_MSG_TOO_BIG_FOR_Q_MGR が返されます。

この方法は，長大メッセージを処理するのに容易な方法です。ただし，使用前に次に示す項目について検討してから使用してください。

- キューマネジャ間の画一性が損なわれます。メッセージデータの最大長は，メッセージが登録される各キュー（転送キューを含む）の MaxMsgLength 属性によって決定されます。しかし，特に転送キューでは，この値はキューマネジャの MaxMsgLength 属性によって制限されることがあります。そのため，リモートキューマネジャにメッセージを転送するときには，メッセージが長過ぎることを判定しにくくなります。
- システムリソースの使用が増加します。例えば，アプリケーションはより大きなバッファを必要とし，共用領域の使用が増加します。長大メッセージが必要な場合にだけキューの格納先が影響されるようにしてください。
- チャネルによるバッチ転送に影響があります。長大メッセージもバッチカウント上では 1 メッセージとして数えられますが，転送に時間が掛かります。そのため，他メッセージの応答時間が増加します。

10.5.2　メッセージのセグメント分割

キューまたはキューマネジャにとってメッセージが長過ぎる場合には，メッセージをセグメント分割できます。セグメントの概要については，「3.6　メッセージグループ」を参照してください。

ここではセグメント分割メッセージの典型的な使用例について説明します。メッセージの登録と取り出し（削除）については，MQPUT 命令と MQGET 命令をトランザクション内で発行します。不完全なグループがネットワーク内でなくなるように，この方法をお勧めします。TP1/Message Queue ではキューマネジャが二相コミットを実行します。

また，取り出しアプリケーションでは，サーバで必要なメッセージまたはセグメントを見つけるのに失敗することがないように，複数のサーバで同じキューを処理し，各サーバは同様のコードを実行します。MQGMO_ALL_MSGS_AVAILABLE または MQGMO_ALL_SEGMENTS_AVAILABLE を事前に指定します。

(1)　キューマネジャによるセグメント分割と組み立て

一つのアプリケーションがメッセージを登録し，もう一方のアプリケーションで取り出すという簡単な例について説明します。メッセージは，アプリケーションが一つのバッファで処理できないほど長くはないが，メッセージが登録されるキューマネジャおよびキューにとっては長過ぎるとします。

登録アプリケーションについての変更点は，キューマネジャが必要に応じてセグメント分割を実行することを許可することです。

```
PMO.Options =（既存のオプション）
MQPUT MD.MsgFlags = MQMF_SEGMENTATION_ALLOWED
```

取り出しアプリケーションでは，セグメント分割されている場合にメッセージをキューマネジャで組み立てるように要求します。

```
GMO.Options = MQGMO_COMPLETE_MSG |（既存のオプション）
MQGET
```

アプリケーションバッファは MQGMO_ACCEPT_TRUNCATED_MSG が指定されていない場合に組み立て後のメッセージを格納するのに十分な長さが必要です。

データ変換が必要な場合には，取り出しアプリケーションで MQGMO_CONVERT を指定して実行してください。これは UOC を使用しない直接的な方法です。送信側チャネルでデータ変換を実行する場合は，メッセージがセグメント分割されていると失敗します。また，完全でないデータはデータ変換を UOC で実行できません。

(2)　アプリケーションによるセグメント分割

アプリケーションによるセグメント分割を使用する場合を次に示します。

1. アプリケーションの一つのバッファで処理するには長過ぎるので，キューマネジャによるセグメント分割だけでは不十分な場合
2. データ変換を送信側チャネルで実行し，データの形式について登録アプリケーションで定義するように，各セグメントの変換が可能なセグメント境界を定義する場合

しかし，データ変換を実行しない場合，または取り出しアプリケーションで常にMQGMO_COMPLETE_MSG を指定する場合には，MQMF_SEGMENTATION_ALLOWED の指定によってキューマネジャによるセグメント分割も許可できます。次に示す例では，メッセージを四つにセグメント分割します。

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

dc_trn_begin()
MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_SEGMENT
MQPUT MD.MsgFlags = MQMF_LAST_SEGMENT
dc_trn_unchained_commit()
```

MQPMO_LOGICAL_ORDER を使用しない場合には，アプリケーションで各セグメントのオフセットと長さを設定してください。この場合，論理状態は自動的に設定されます。

取り出しアプリケーションでは，組み立て後のメッセージ全体を格納する領域は不要です。ただし，セグメントの個別な処理が必要です。

セグメント分割されたメッセージについては，論理メッセージを構成する全セグメントがそろうまでは，セグメントの処理をアプリケーションで開始したくないことがあります。その場合は，MQGMO_ALL_SEGMENTS_AVAILABLE を最初のセグメントに指定できます。MQGMO_LOGICAL_ORDER を指定する場合に，現在の論理メッセージがあるときには，MQGMO_ALL_SEGMENTS_AVAILABLE は無視されます。

論理メッセージの最初のセグメントをいったん取り出したら，論理メッセージの残りのセグメントを順に取り出すために MQGMO_LOGICAL_ORDER を指定できます。

異なるグループ内のメッセージについての配慮は不要です。そのようなメッセージが発生した場合は，キューで各メッセージの最初のセグメントが出現する順に処理されます。

```
GMO.Options = MQGMO_SYNCPOINT | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_SEGMENTS_AVAILABLE | MQGMO_WAIT
dc_trn_begin()
while ( SegmentStatus == MQSS_SEGMENT ) {
  MQGET
  /* 論理メッセージの残りのセグメントを処理 */
  ...
}
dc_trn_unchained_commit()
```

(3) アプリケーションによる論理メッセージのセグメント分割

メッセージのグループ内での論理的順序を保持する必要があり，かつ，メッセージがアプリケーションによるセグメント分割を必要とするほど長大である場合について説明します。

ここでは，四つの論理メッセージグループを登録します。三つのメッセージが長大であり，登録アプリケーションによるセグメント分割を必要とします。

```
PMO.Options = MQPMO_LOGICAL_ORDER | MQPMO_SYNCPOINT

dc_trn_begin()

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

dc_trn_unchained_commit()
```

取り出しアプリケーションでは，MQGMO_ALL_MSGS_AVAILABLE を最初の MQGET 命令に指定します。これはグループ全体が利用できるようになるまではグループのメッセージもセグメントも取り出さないようにする指定です。グループの最初の物理メッセージを取り出したら，グループの残りのセグメントとメッセージを順に取り出すために MQGMO_LOGICAL_ORDER を使用できます。

```
GMO.Options = MQGMO_SYNCPOINT | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_MSGS_AVAILABLE | MQGMO_WAIT

dc_trn_begin()
while ( (GroupStatus != MQGS_LAST_MSG_IN_GROUP) ||
          (SegmentStatus != MQSS_LAST_SEGMENT) ) {
  MQGET
  /* セグメントまたは完全な論理メッセージの処理
     返却値を調べるにはGroupStatusとSegmentStatusを使用
     してください。 */
  ...
}
  dc_trn_unchained_commit()
```

注意

MQGMO_LOGICAL_ORDER 指定時に現在のグループがある場合には，MQGMO_ALL_MSGS_AVAILABLE は無視されます。

(4) 複数トランザクションにわたるセグメント分割メッセージの登録と取り出し

複数トランザクションにわたるセグメント分割メッセージの登録と取り出しは，グループ化したメッセージの処理の場合と同様です。詳細については，「10.3.2(2) 複数トランザクションにわたるグループの登録と取り出し」を参照してください。

MQMF_SEGMENTATION_ALLOWED を指定するメッセージの登録，および MQGMO_COMPLETE_MSG を指定するメッセージの取り出しは，トランザクション内で実行してください。トランザクション外で実行した場合，エラーが発生します。

10.5.3 参照メッセージ

参照メッセージを使用すると，送信元および送信先のノードにあるキューにオブジェクトを格納しなくても，ノード間で長大オブジェクトを転送できます。メールアプリケーションなど他形式の既存データがある場合には特に有効です。

この場合，TP1/Message Queue では，送信側および受信側でメッセージ編集出口 UOC が必要です。メッセージ編集出口 UOC については，マニュアル「TP1/Message Queue 使用の手引」を参照してください。

(1) MQRMH 構造体と MQMD 構造体の使用

MQMD 構造体の Format フィールドに MQFMT_REF_MSG_HEADER を設定してください。MQHREF フォーマットは，MQGET 命令で要求されるときに TP1/Message Queue によって自動的に後続のデータとともに変換されます。

ここでは，MQRMH 構造体の DataLogicalOffset および DataLogicalLength フィールドの使用例について説明します。

登録アプリケーションは，参照メッセージを次に示す状態で登録します。

- 物理データなし
- DataLogicalOffset ＝ 0（このメッセージはオブジェクト全体を表します）
- DataLogicalLength ＝ 0

オブジェクトが 70000 バイトの長さであり，メッセージ編集出口 UOC が最初の 40000 バイトをチャネルを経由して送信するとき，参照メッセージには次に示すデータが含まれます。

- MQRMH 構造体に続く 40000 バイトの物理データ
- DataLogicalLength ＝ 40000
- DataLogicalOffset ＝ 0（オブジェクトの先頭から）

その後，登録アプリケーションが追加メッセージを転送キューに登録するとき，次に示すデータが含まれます。

- 物理データなし
- DataLogicalLength ＝ 0（オブジェクトの終わりまで）。30000 を指定することもできます。
- DataLogicalOffset ＝ 40000（開始位置）

メッセージ編集出口 UOC が残りの 30000 バイトを追加して送信するとき，参照メッセージには次に示すデータが含まれます。

- MQRMH 構造体に続く 30000 バイトの物理データ
- DataLogicalLength ＝ 30000
- DataLogicalOffset ＝ 40000（開始位置）

MQRMHF_LAST も設定されます。

10.6　メッセージの待ち合わせ

メッセージがキューに到着するまでアプリケーションで待ち合わせしたい場合は，MQGMO 構造体の Options フィールドに MQGMO_WAIT を指定してください。MQGET 命令の待ち合わせ最大時間（ミリ秒）は，MQGMO 構造体の WaitInterval フィールドに設定します。

待ち合わせ最大時間内にメッセージが到着しない場合，MQGET 命令は MQRC_NO_MSG_AVAILABLE の理由コードで完了します。

WaitInterval フィールドに MQWI_UNLIMITED を指定すると，無制限に待つことができます。しかし，ユーザの制御外にあるイベントによってアプリケーションが長く待たされることもあるので，この定数を使用する場合は注意してください。また，無制限に待ち合わせている間にアプリケーションまたは OpenTP1 を終了させた場合，MQGET 命令は理由コード MQRC_Q_MGR_STOPPING で完了します。

注意

複数のアプリケーションが同じ共用キューについてメッセージの削除を待ち合わせる場合，メッセージの到着が有効になるアプリケーションは一つだけです。しかし，複数のアプリケーションがメッセージの検索を待ち合わせる場合，すべてのアプリケーションで有効になります。

待ち合わせ最大時間が経過する前に，キューまたはキューマネジャの状態が変わる場合は，次に示すとおりに動作します。

- キューマネジャが強制停止させられると，MQGET 命令は MQRC_Q_MGR_STOPPING または MQRC_CONNECTION_BROKEN の理由コードで完了します。
- キューの属性（別名キューの属性を含む）が変更されて取り出しが禁止された場合，待ち合わせは中断され，MQGET 命令は MQRC_GET_INHIBITED の理由コードで完了します。

10.7　制御情報とアプリケーションデータの変換

アプリケーションがメッセージをキューに登録するとき，ローカルキューマネジャは制御情報をメッセージ記述子に追加し，キューマネジャと MCA でメッセージを処理しやすくします。通常は，ローカルシステムの環境の文字セットとマシンコード形式でメッセージヘッダのデータフィールドを作成します。

TP1/Message Queue では，MQA サービス定義の mqa_local_ccsid オペランドを指定することによって，キューマネジャがメッセージ記述子に設定する文字セットを指定できます。

メッセージをシステム間で移動するとき，必要に応じて，受信側システムで必要な文字セットとマシンコード形式にアプリケーションデータを変換できます。この変換は受信側システムのアプリケーション，または送信側システムの MCA で実行できます。受信側システムでデータ変換がサポートされている場合は，送信側システムで変換してから送信する方法よりも，受信側アプリケーションでアプリケーションデータを変換する方法をお勧めします。

MQGET 命令に指定する MQGMO 構造体の Options フィールドに MQGMO_CONVERT を指定するときにアプリケーションデータは変換されます。このときの条件を次に示します。

- メッセージに対応する MQMD 構造体の CodedCharSetId および Encoding フィールドが，MQGET 命令に指定した MQMD 構造体の値とは異なること。
- メッセージに関連する MQMD 構造体の Format フィールドが MQFMT_STRING であること。
- MQGET 命令に指定された BufferLength パラメタが 0 でないこと。
- メッセージデータ長が 0 でないこと。
- メッセージに関連する MQMD 構造体の CodedCharSetId および Encoding フィールドと，MQGET 命令に指定した MQMD 構造体の値の変換をキューマネジャがサポートすること。
- キューマネジャがメッセージフォーマットの変換をサポートすること。メッセージに関連する MQMD 構造体の Format フィールドが組み込みフォーマットの場合は，キューマネジャはメッセージを変換できます。組み込みフォーマットでない場合は，データ変換を実行する UOC をユーザが作成しなくてはなりません。

MCA でデータ変換を実行する場合には，MQT 通信構成定義の mqtalccha 定義コマンドの-d オプションの cnvccsid オペランドを指定してください。データ変換に失敗すると，メッセージは送信側キューマネジャのデッドレターキューに登録され，MQDLH 構造体の Feedback フィールドに理由が設定されます。メッセージをデッドレターキューに登録できない場合は，チャネルは終了し，無変換のメッセージが転送キューに残ります。送信側 MCA ではなく受信側アプリケーションでデータ変換する場合は，これを回避できます。

通常，組み込みフォーマットやデータ変換の UOC は，文字データとして記述されるデータを，メッセージで使用される文字セットから必要な文字セットに変換します。また，数値データとして記述されるデータを必要なマシンコード形式に変換します。

TP1/Message Queue がサポートする文字セットおよびマシンコード形式については，マニュアル「TP1/Message Queue プログラム作成リファレンス」を参照してください。

10.8　キューのメッセージの検索

キューのメッセージを検索するために MQGET 命令を使用するには，次に示すとおりに操作します。

1. MQOO_BROWSE を指定して MQOPEN 命令でキューを検索用にオープンします。
2. キューの最初のメッセージを検索するには，MQGMO_BROWSE_FIRST を指定して MQGET 命令を発行します。必要なメッセージを検索するために，一とおりのメッセージを参照するには，MQGMO_BROWSE_NEXT を指定して MQGET 命令を繰り返し発行します。

   すべてのメッセージを参照するには，各 MQGET 命令のあとで MQMD 構造体の MsgId と CorrelId フィールドをヌルに設定してください。
3. MQCLOSE 命令を発行してキューをクローズします。

10.8.1　検索カーソル

検索のためにキューをオープンするとき，命令は検索オプションの MQGET 命令で使用するための検索カーソルを生成します。検索カーソルは，キューの最初のメッセージの前に位置づけられる論理的なポインタです。

同じキューに対して MQOPEN 命令を複数発行することによって，一つのアプリケーションで複数の検索カーソルを有効にできます。

検索用に MQGET 命令を発行するときには，MQGMO 構造体には次に示すオプションを指定してください。

MQGMO_BROWSE_FIRST
　MQMD 構造体で指定された条件を満たす最初のメッセージを取り出します。

MQGMO_BROWSE_NEXT
　MQMD 構造体で指定された条件を満たす次のメッセージを取り出します。

どちらの場合も，メッセージはキューに残ります。

キューをオープンするとき，検索カーソルはキューで最初のメッセージの直前に位置づけられます。そのため，MQOPEN 命令の直後に MQGET 命令を発行する場合は，最初のメッセージを検索するのに MQGMO_BROWSE_NEXT を指定できます。MQGMO_BROWSE_FIRST を指定しなくてもかまいません。

キューからメッセージが取り出される順序は，キューの MsgDeliverySequence 属性によって決定されます。詳細については，「10.3　メッセージがキューから取り出される順序」を参照してください。

(1)　FIFO 順のキュー

FIFO 順でキューにある最初のメッセージは，最も長い時間キューにあるメッセージです。

キューのメッセージを順に読み込むには，MQGMO_BROWSE_NEXT を使用してください。この順でキューの最後までメッセージを持っている場合には，任意のメッセージを検索できます。キューの最後に届いたとき，検索カーソルはその位置で止まり MQRC_NO_MSG_AVAILABLE を返します。追加のメッセージが到着することを待ってそのままにしたり，または MQGMO_BROWSE_FIRST の指定でリセットしてキューの最初に移動させたりしてください。

(2) 優先度順のキュー

優先度順でキューにある最初のメッセージは，最も長い時間キューにあるメッセージで，MQOPEN 命令の発行時に最も高い優先度を指定されたメッセージです。

キューのメッセージを読み込むには，MQGMO_BROWSE_NEXT を使用してください。

検索カーソルは次のメッセージを指します。最高の優先度のメッセージから始まり，最低の優先度のメッセージで終わります。現在の検索カーソルによって識別されるメッセージと同等またはそれ以下のメッセージが続く間は，キューに登録されたメッセージを検索します。

キューに登録された優先度の高いメッセージが検索されるのは，次に示す場合だけです。

- 再度の検索のためにキューをオープンするときに，新規の検索カーソルが生成される場合
- MQGMO_BROWSE_FIRST を使用する場合

(3) コミット待ちメッセージ

コミット待ちメッセージは，検索時に検出されません。検索カーソルは読み飛ばします。トランザクション内のメッセージは，コミットされるまで検索されません。コミット時もメッセージのキューでの位置は変わらないため，読み飛ばされたコミット待ちメッセージはコミット後も読めません。読み直すには，MQGMO_BROWSE_FIRST を使用します。

10.8.2 メッセージ長が不明なときの検索

メッセージ長が不明なときにメッセージを検索し，かつ，MsgId，CorrelId，または GroupId フィールドを使用したくない場合には，MQGMO_BROWSE_MSG_UNDER_CURSOR を使用できます。

手順を次に示します。

1. MQGET 命令を次に示す状態で発行します。
    - MQGMO_BROWSE_FIRST または MQGMO_BROWSE_NEXT を指定
    - MQGMO_ACCEPT_TRUNCATED_MSG を指定
    - バッファ長に 0 を指定

注意

他アプリケーションで同じメッセージを取り出す場合には，MQGMO_LOCK オプションの使用も検討してください。MQRC_TRUNCATED_MSG_ACCEPTED が返されるようにしてください。

2. 必要な領域を確保するために，MQGET 命令で返された DataLength パラメタを使用してください。

3. MQGMO_BROWSE_MSG_UNDER_CURSOR で MQGET 命令を発行してください。

検索カーソルで示されるメッセージは，取り出された最後のメッセージです。検索カーソルは移動しません。ユーザは MQGMO_LOCK を使用してメッセージを排他状態にしたり，MQGMO_UNLOCK で排他状態を解除したりできます。

キューのオープン後に MQGMO_BROWSE_FIRST または MQGMO_BROWSE_NEXT の MQGET 命令が発行されていない場合は，命令は失敗します。

10.8.3 検索済みメッセージの削除

メッセージの検索と同時に削除の目的でキューをオープンした場合には，ユーザは検索済みのメッセージをキューから削除できます。MQOPEN 命令には MQOO_BROWSE と同時に MQOO_INPUT_*を指定してください。

メッセージを削除するには，MQGET 命令を再度発行してください。MQGMO 構造体の Options フィールドには MQGMO_MSG_UNDER_CURSOR を指定します。この場合，MQGET 命令は MQMD 構造体の MsgId，CorrelId，および GroupId フィールドを無視します。

検索と削除の間に，ユーザの検索カーソルがあるメッセージを他アプリケーションが削除することもあります。この場合，メッセージが利用できないという理由コードが MQGET 命令から返されます。

10.9　論理的順序でのメッセージ検索

キューにあるメッセージの順序には，論理的順序と物理的順序があります。詳細については，「10.3.2　論理的順序と物理的順序」を参照してください。この順序はメッセージの検索時には特に重要です。一般的には，メッセージは削除されないで，検索は必ずしもキューの最初から開始しないためです。アプリケーションで一つのグループの複数のメッセージを論理的順序で一とおり検索する場合には，次のグループの最初のメッセージが後続するようにしてください。一つのグループの最後のメッセージが，物理的には次のグループの最初のメッセージのあとになる可能性があります。MQGMO_LOGICAL_ORDER はキューを調べるときに論理的順序に従うことを指定します。

検索時の MQGMO_ALL_MSGS_AVAILABLE または MQGMO_ALL_SEGMENTS_AVAILABLE は注意して使用してください。ここでは，MQGMO_ALL_MSGS_AVAILABLE の論理メッセージについて説明します。このオプションの機能は，グループ内のメッセージの残りがすべてある場合にだけ論理メッセージを利用できるようにすることです。残りがそろわない場合にはメッセージは渡されません。つまり，転送されなかったメッセージがあとから到着したときには MQGMO_BROWSE_NEXT で通知されません。

例えば，次に示す論理メッセージがあるとします。

```
グループ123の論理メッセージ1（最後でない）
グループ456の論理メッセージ1（最後でない）
グループ456の論理メッセージ2（最後）
```

ここで MQGMO_ALL_MSGS_AVAILABLE の検索が発行されたとします。このときグループ 456 の論理メッセージ 1 が返され，検索カーソルはこの論理メッセージ上に残ります。そして，グループ 123 の論理メッセージ 2（最後）が到着したとします。

```
グループ123の論理メッセージ1（最後でない）
グループ123の論理メッセージ2（最後）
グループ456の論理メッセージ1（最後でない）　<== 検索カーソル
グループ456の論理メッセージ2（最後）
```

ここで MQGMO_BROWSE_NEXT の検索が発行されても，グループ 123 がそろっていることは検出されません。グループの最初のメッセージが検索カーソルの前にあるからです。

グループが完全なときにメッセージを削除しながら取り出す場合などには，MQGMO_ALL_MSGS_AVAILABLE を MQGMO_BROWSE_FIRST とともに使用できます。しかし，読み飛ばした到着メッセージを検出するには，検索を繰り返さなければなりません。MQGMO_WAIT を MQGMO_BROWSE_NEXT および MQGMO_ALL_MSGS_AVAILABLE とともに発行するだけでは不十分です。同様のことは，メッセージの検索後に，より高い優先度のメッセージが到着した場合にも発生します。

10.9.1　グループ内のメッセージ検索

ここでは，キューの各メッセージを論理的順序で一とおり検索する例を示します。

キューのメッセージはグループ化されていたり，されていなかったりします。グループ化されたメッセージには，グループ内のすべてのメッセージが到着するまで，アプリケーションでグループの処理を開始させたくないことがあります。そのときは MQGMO_ALL_MSGS_AVAILABLE をグループの最初のメッセージに指定します。グループ内の以降のメッセージには，このオプションは不要です。

この例では MQGMO_WAIT を使用します。待ち合わせは新規グループが到着すると完了します。ただし，検索カーソルがグループ内の最初の論理メッセージを過ぎていると残りのメッセージが到着しても，完了しません。また，一定間隔で待ち合わせることによって，新規メッセージまたはセグメントを待つ間にアプリケーションがループに入ったままにならないようにできます。

論理的順序の検索を指定するために，最初から最後まで MQGMO_LOGICAL_ORDER を使用します。これはメッセージを削除する MQGET 命令の例とは異なる点です。メッセージを削除する MQGET 命令の例では各グループが削除されるので，グループ内の最初または唯一のメッセージを検索するときに MQGMO_LOGICAL_ORDER は使用しません。

アプリケーションのバッファは，メッセージのセグメント分割の有無に関係なく，メッセージ全体を保持するのに十分な大きさがあるとします。そのため MQGET 命令には MQGMO_COMPLETE_MSG を指定します。

グループ内の論理メッセージの検索の例について，次に示します。

```
/* グループ内の最初のメッセージ,
   またはグループにないメッセージの検索 */
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
/* 最初または唯一のメッセージの確認 */
...
GMO.Options = MQGMO_BROWSE_NEXT | MQGMO_COMPLETE_MSG
            | MQGMO_LOGICAL_ORDER
while ( GroupStatus == MQGS_MSG_IN_GROUP ) {
  MQGET
  /* グループ内の残りのメッセージの確認 */
  ...
}
```

上記の検索は MQRC_NO_MSG_AVAILABLE が返されるまで繰り返されます。

10.9.2　削除しながらの取り出し

ここでは，グループを削除しながら取り出すか決定する前に，アプリケーションでグループ内の各論理メッセージを検索する例を示します。

この例ではグループ全体の検索が終わると，最初に戻ってメッセージを削除しながら取り出します。

この例では各グループを削除するので，グループ内の最初または唯一のメッセージを検索するときには，MQGMO_LOGICAL_ORDER は使用できません。

検索と削除しながらの取り出しの例について，次に示します。

```
GMO.Options = MQGMO_BROWSE_NEXT | MQGMO_COMPLETE_MSG
            | MQGMO_LOGICAL_ORDER
            | MQGMO_ALL_MSGS_AVAILABLE | MQGMO_WAIT
while ( GroupStatus == MQGS_MSG_IN_GROUP ) {
  MQGET
  /* グループ内の残りのメッセージを調査
      （または削除しながら取り出すか決定するのに
       必要なだけ調査）*/
  ...
}

if ( 削除しながらグループを取り出す )
  if ( GroupStatus == ' ' )
    /* グループ化されないメッセージの取り出し */
    GMO.Options = MQGMO_MSG_UNDER_CURSOR | MQGMO_SYNCPOINT
    MQGET GMO.MatchOptions = 0
    /* メッセージの処理 */
    ...
  else
    /* 一つ以上のメッセージを取り出しました。検索カーソルは通常は
       グループの最初にはありません。そのため，GroupIdを設定し，
       MsgSeqNumberに1を設定します。
       グループ化の有無に影響されないほかの方法では，検索時に
       最初のメッセージのMsgIdを保存し，それを設定します。 */
    GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT
    MQGET GMO.MatchOptions = MQMO_MATCH_GROUP_ID
                           | MQMO_MATCH_MSG_SEQ_NUMBER,
          (MQMD.GroupId = メッセージ記述子に既存の値)
           MQMD.MsgSeqNumber = 1
    /* 最初または唯一のメッセージの処理 */
    ...
    GMO.Options = MQGMO_COMPLETE_MSG | MQGMO_SYNCPOINT
                | MQGMO_LOGICAL_ORDER
    while ( GroupStatus == MQGS_MSG_IN_GROUP ) {
      MQGET
      /* グループ内の残りメッセージの処理 */
      ...
    }
```

10.10 MQGET 命令が失敗する状況

メッセージを取り出したいキューに取り出しが許可されていない場合には，MQGET 命令は失敗し，MQRC_GET_INHIBITED の理由コードを返します。検索用に MQGET 命令を使用する場合にもこのことは発生します。他アプリケーションからキューの属性を定期的に変更するように設計しているときなどには，あとから実行することによってメッセージの取り出しを成功させることができます。

動的キュー（一時的動的キューまたは永続的動的キュー）が削除された場合，取得済みのオブジェクトハンドルを使用している MQGET 命令は失敗し，MQRC_Q_DELETED の理由コードを返します。

11 オブジェクト属性の照会と設定

この章では，オブジェクト属性の照会と設定について説明します。

11.1　オブジェクト属性の照会と設定の概要

属性はオブジェクトの性質を定義します。属性は，キューマネジャがオブジェクトを処理する方法に影響します。各オブジェクトの属性については，マニュアル「TP1/Message Queue プログラム作成リファレンス」を参照してください。

オブジェクトの属性は定義時に設定されますが，あとから運用コマンドで変更できる属性もあります。

MQINQ 命令を使用するとこれらの属性の現在の値を照会できます。MQSET 命令を使用すると幾つかのキュー属性を変更できます。キュー以外のオブジェクトの属性は MQI 命令では変更できません。

MQINQ 命令と MQSET 命令では，照会または設定したい属性を識別するためにセレクタの配列を使用します。操作可能な各属性にセレクタがあります。セレクタの名前には，種類ごとのプリフィクスがあります。

MQCA_
　文字データ（キュー名など）の属性を表すセレクタ

MQIA_
　数値（メッセージ登録数など）または定数値（トランザクション有効性など）の属性を表すセレクタ

MQINQ 命令または MQSET 命令を使用する前には，アプリケーションをキューマネジャに接続し，属性の照会または設定を実行したいオブジェクトを MQOPEN 命令でオープンしてください。これらの操作については，「7.　キューマネジャの接続と切り離し」および「8.　オブジェクトのオープンとクローズ」を参照してください。

11.2 オブジェクト属性の照会

任意のオブジェクト属性について，MQINQ 命令で照会できます。

MQINQ 命令の入力として，次に示す項目を指定してください。

- コネクションハンドル（HCONN）
- オブジェクトハンドル（Hobj）
- セレクタ数
- 属性セレクタの配列

  各セレクタは MQCA_*または MQIA_*の形式です。各セレクタは照会したい属性を表し，オブジェクトハンドルに示すオブジェクトタイプにとって有効でなければなりません。セレクタの指定順序は任意です。
- 照会する整数型属性の数

  整数型属性を照会しない場合は 0 を指定してください。
- 文字型属性値の長さ

  各文字型属性値を保持するのに必要な長さの合計以上を指定してください。文字型属性を照会しない場合は 0 を指定してください。

MQINQ 命令の出力を次に示します。

- 配列にコピーされる一連の整数型属性値

  値の数は IntAttrCount パラメタで指定されます。IntAttrCount または SelectorCount パラメタが 0 の場合はこのパラメタは使用されません。
- 理由コード

  部分的完了状態が発生する場合は三つあります。
  - セレクタがキュータイプに適合しない場合
  - 文字型属性値に対して十分な領域がない場合
  - 整数型属性値に対して十分な領域がない場合

  複数発生するときには，最初に適用されたものが返されます。

キューを登録用または照会用にオープンして，ローカルでないクラスタキューに解決される場合，ユーザが照会できるのはキュー名，キュータイプ，および共通属性だけです。共通属性の値は MQOO_BIND_ON_OPEN を指定して選択されたキューの値です。MQOO_BIND_NOT_FIXED を指定した場合，またはキューの DefBind 属性が MQBND_BIND_NOT_FIXED のときに MQOO_BIND_AS_Q_DEF を指定した場合は，解決可能な任意のクラスタキューの値です。

注意

MQINQ 命令が返す値は，選択した属性の一時的な値です。返された値に従ってアプリケーションが動作する前に変更されることがあります。

11.3 MQINQ 命令が失敗する状況

属性を照会するために別名キューをオープンする場合は，ベースキューではなく別名キューの属性だけが返されます。しかし，別名キューが解決されるベースキューの定義がキューマネジャによってオープンされ，自アプリケーションによる MQOPEN 命令と MQINQ 命令の間に他アプリケーションがベースキューの使用方法を変更する場合は，自アプリケーションの MQINQ 命令は失敗し，MQRC_OBJECT_CHANGED の理由コードが返されます。別名キューの属性が変更される場合も命令は失敗します。

同様に，属性の照会用にリモートキューをオープンするとき，リモートキューのローカル定義の属性だけが返されます。

属性を照会するキュータイプに有効でない一つ以上のセレクタを指定する場合，MQINQ 命令は警告で完了し，次に示す出力を設定します。

- 整数型属性について，IntAttrs パラメタの該当する要素に MQIAV_NOT_APPLICABLE
- 文字型属性について，CharAttrs パラメタの該当する要素にアスタリスク

属性を照会するオブジェクトタイプに有効でない一つ以上のセレクタを指定する場合，MQINQ 命令は失敗し MQRC_SELECTOR_ERROR の理由コードを返します。

モデルキューを照会するために MQINQ 命令は使用できません。TP1/Message Queue では，mqainq コマンドを使用します。

11.4　キュー属性の設定

MQSET 命令を使用する場合は，次に示すキュー属性だけを設定できます。

- InhibitGet（リモートキューは除く）
- DistLists
- InhibitPut
- TriggerControl
- TriggerType
- TriggerDepth
- TriggerMsgPriority
- TriggerData

MQSET 命令は MQINQ 命令と同じパラメタを取ります。ただし，MQSET 命令では，完了コードと理由コード以外はすべて入力パラメタです。部分的完了状態はありません。

注意

ローカルで定義されたキュー以外のオブジェクトの属性は，MQSET 命令で変更できません。

12 コミットとロールバック

この章では，コミットとロールバックについて説明します。

12.1　コミットとロールバックの概要

コミットとロールバックの概要について説明します。

12.1.1　コミットとロールバックの性質

トランザクション内でアプリケーションがメッセージをキューに登録する場合，そのメッセージは，アプリケーションでトランザクションをコミットするときに他アプリケーションから参照できるようになります。コミットするまでは他アプリケーションから参照できません。また，コミット完了前のメッセージは，キューのメッセージ登録数（CurrentQDepth 属性）としてカウントされます。そのため，コミット完了前のメッセージ登録数（CurrentQDepth 属性）が，最大メッセージ登録数（MaxQDepth 属性）と等しくなる場合，他アプリケーションからの MQPUT 命令は MQRC_Q_FULL でエラーとなります。トランザクションをコミットすると，すべての変更についてデータの一貫性が保たれます。アプリケーションでエラーを検出したために登録を取り消したい場合には，トランザクションをロールバックできます。ロールバックを実行すると，キューにメッセージは登録されません。TP1/Message Queue がコミットとロールバックを実行する方法は，OpenTP1 システムに従います。

また，トランザクション内でメッセージをキューから取り出す場合，アプリケーションがトランザクションをコミットするまでキューにメッセージが残ります。しかし，他アプリケーションでメッセージを取り出すことはできません。アプリケーションがトランザクションをコミットするときにメッセージは削除されます。コミット完了前のメッセージは，キューのメッセージ登録数（CurrentQDepth 属性）としてカウントします。アプリケーションがトランザクションをロールバックする場合，TP1/Message Queue は，アプリケーションによる取り出し操作前の状態にキューを戻します。

MQSET 命令または運用コマンドによるキュー属性の変更は，トランザクションのコミットとロールバックによる影響を受けません。

12.1.2　同期点の取得とトランザクション

データの一貫性を保持してトランザクションをコミットまたはロールバックする処理を，同期点の取得といいます。

変更操作をコミットするかロールバックするかは，通常，処理の最後に決定します。しかし，処理内部の論理的なポイントでデータ変更の同期を取れた方がアプリケーションにとって便利なことがあります。この論理的なポイントを**同期点**といい，同期点と同期点の間での変更処理を**トランザクション**といいます。一つのトランザクションには複数の MQGET 命令と MQPUT 命令を参加させることができます。

12.1.3　単相コミット

アプリケーションが実行したキューへの変更をほかのリソースマネジャと連携しないで処理することを**単相コミット**といいます。

12.1.4　二相コミット

アプリケーションが実行したキューへの変更をほかのリソースマネジャ（データベースなど）と連携して処理することを**二相コミット**といいます。

この処理ではすべてのリソースについての変更は同時にコミットまたはロールバックされます。

トランザクションの処理を補助するために，TP1/Message Queue は MQMD 構造体の BackoutCount フィールドを提供します。このフィールドは，トランザクション内でメッセージがロールバックされるたびに値が増えます。詳細については，マニュアル「TP1/Message Queue プログラム作成リファレンス」を参照してください。

12.2 TP1/Message Queue のトランザクション

TP1/Message Queue は，二相コミットに対応します。単相コミットの機能は提供しません。

アプリケーションでは登録操作と取り出し操作についてトランザクションへの参加を指定できます。登録操作では，MQPMO 構造体の Options フィールドに MQPMO_SYNCPOINT を指定してください。取り出し操作では，MQGMO 構造体の Options フィールドに MQGMO_SYNCPOINT を指定してください。明示的に指定しない場合，TP1/Message Queue では，命令をトランザクション内から呼び出すかどうかで，トランザクションへの参加と不参加が決定されます。各オプション指定時の動作については，マニュアル「TP1/Message Queue プログラム作成リファレンス」を参照してください。

トランザクションへの参加と不参加の例を次に示します。

```
MQPUT(MQPMO_NO_SYNCPOINTを指定) /* トランザクションに参加しない */
MQGET(MQGMO_NO_SYNCPOINTを指定) /* トランザクションに参加しない */
dc_trn_begin()                  /* トランザクションの開始 */
MQGET(MQGMO_SYNCPOINTを指定)    /* トランザクションに参加 */
MQGET(MQGMO_NO_SYNCPOINTを指定) /* トランザクションに参加しない */
MQPUT(MQPMO_SYNCPOINTを指定)    /* トランザクションに参加 */
MQPUT(MQPMO_NO_SYNCPOINTを指定) /* トランザクションに参加しない */
dc_trn_unchained_commit()       /* 同期点の取得
                                   （トランザクションの終了） */
```

なお，トランザクション制御用オプションをリンケージしないで UAP を作成する場合，次の点に注意してください。

- MQPUT 命令および MQPUT1 命令

  MQPMO 構造体の Options パラメタに MQPMO_SYNCPOINT が指定され，トランザクション内で MQPUT 命令または MQPUT1 命令が発行された場合，MQRC_SYNCPOINT_NOT_AVAILABLE でエラーリターンします。

  MQPMO 構造体の Options パラメタに MQPMO_SYNCPOINT および MQPMO_NO_SYNCPOINT が省略され，トランザクション内で MQPUT 命令または MQPUT1 命令が発行された場合，MQRC_SYNCPOINT_NOT_AVAILABLE でエラーリターンします。

- MQGET 命令

  MQGMO 構造体の Options パラメタに MQGMO_SYNCPOINT が指定され，トランザクション内で MQGET 命令が発行された場合，MQRC_SYNCPOINT_NOT_AVAILABLE でエラーリターンします。

  MQGMO 構造体の Options パラメタに MQGMO_SYNCPOINT および MQGMO_NO_SYNCPOINT が省略され，トランザクション内で MQGET 命令が発行された場合，MQRC_SYNCPOINT_NOT_AVAILABLE でエラーリターンします。

- MQCLOSE 命令および MQDISC 命令

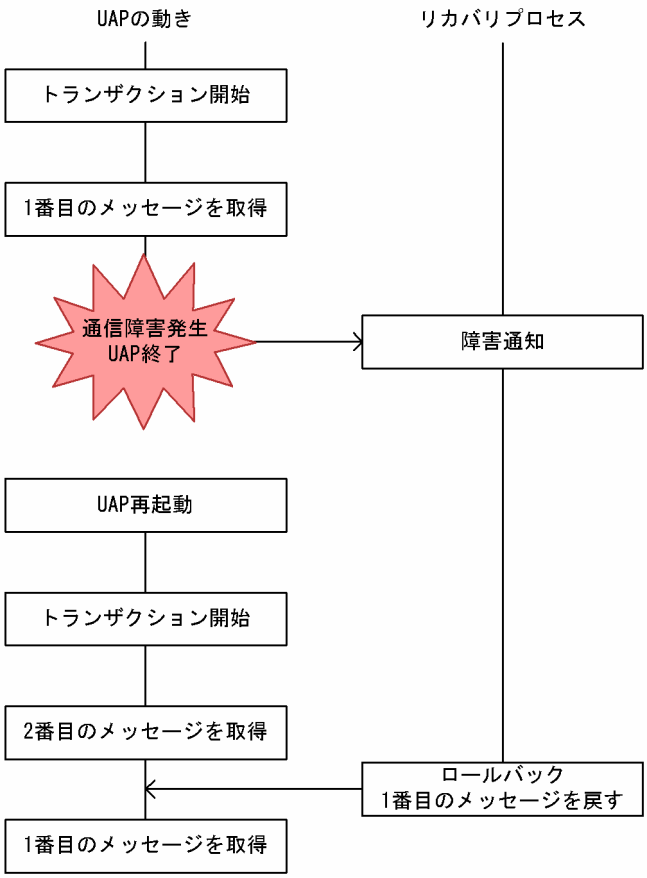
  MQCLOSE 命令または MQDISC 命令が発行されない状態でユーザアプリケーションが正常終了した場合，使用していた資源が解放されないことがあります。

12.3　注意事項

OpenTP1 の同期点処理は通信障害，リソースマネジャ障害，UAP 障害などの要因によって，UAP とは別プロセス（リカバリプロセス）で並行してロールバック処理が行われる場合があります。そのため，キューからメッセージを取り出す UAP が一つであっても，ロールバック処理と並行して UAP が動作した場合，メッセージの順序性が保証できないことがあります。

メッセージの順序性が失われる例を次の図に示します。

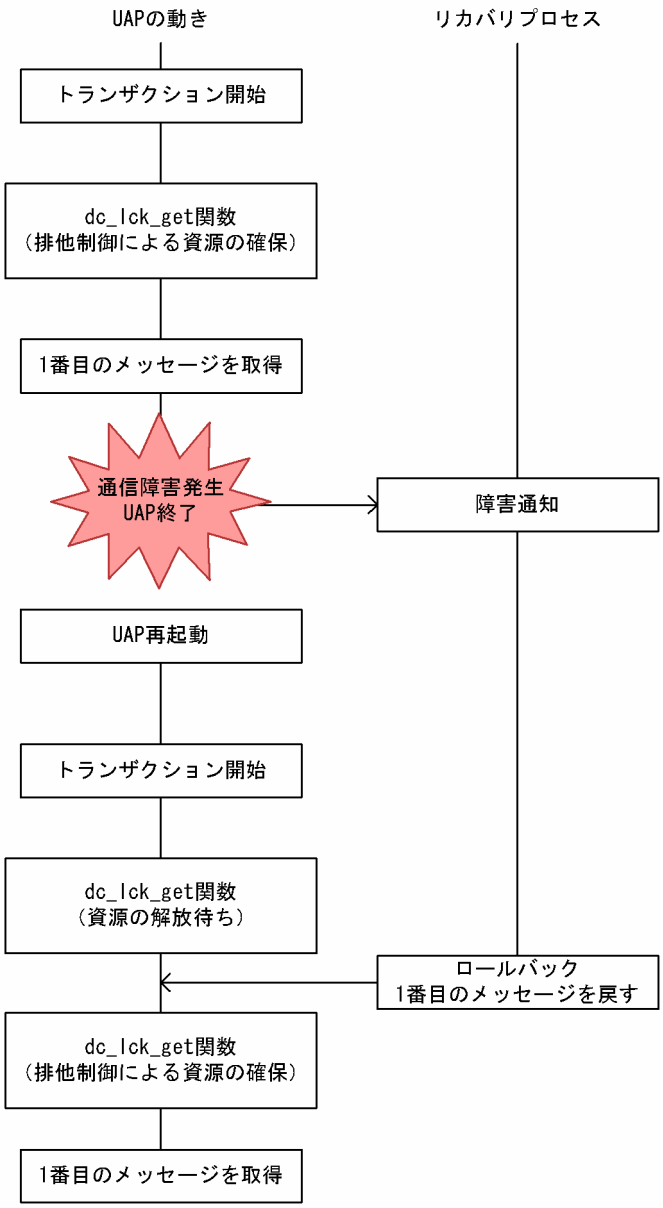
図 12-1　メッセージの順序性が失われる例



メッセージの順序性を保証するためには，メッセージの取り出しを行う UAP が別プロセスで行われるロールバック処理を待ち合わせる必要があります。例えば dc_lck_get 関数を呼び出して UAP で使用する資源の排他をグローバルトランザクション単位で確保することでメッセージの順序性を確保できます。

dc_lck_get 関数による資源の排他要求の例を次の図に示します。

図 12-2　dc_lck_get 関数による資源の排他要求の例



13 トリガによるアプリケーション開始

この章では，トリガによるアプリケーション開始について説明します。

13.1　トリガによるアプリケーション開始の概要

キューを処理するアプリケーションを常時動作させると，メッセージがキューに到着したときにいつでも取り出せます。ただし，メッセージがいつキューに到着するのかわからないと不便なことがあります。例えば，取り出すメッセージがないときにもシステムリソースを消費してしまいます。

そこで，取り出せるメッセージがあるときにアプリケーションを自動的に開始するための機能を提供します。この機能をトリガといいます。

13.2　トリガの概要

キューマネジャは，トリガイベントが発生する条件を定義します。キューについてトリガが有効なときにトリガイベントが発生すると，キューマネジャは**イニシエーションキュー**に**トリガメッセージ**を送信します。イニシエーションキューにトリガメッセージがあることはトリガイベントの発生を意味します。

キューマネジャが生成するトリガメッセージは，永続メッセージではありません。そのため，ログ取得が減り，再度開始するときの複製動作が最小になることから，再度開始するときの時間が短縮されます。

イニシエーションキューを処理するアプリケーションを**トリガモニタアプリケーション**といいます。トリガモニタアプリケーションは，トリガメッセージを読み込み，トリガメッセージにある情報に基づいて適切な動作を実行します。通常は，他アプリケーションを開始し，トリガメッセージ発生の原因となったキューを処理します。トリガモニタアプリケーションは，イニシエーションキューからメッセージを読み込むアプリケーションです。

キューでトリガを有効にする場合，必要に応じて，キューに対応するプロセス定義オブジェクトを作成できます。プロセス定義オブジェクトには，トリガイベント発生の原因となったメッセージを処理するアプリケーションについての情報があります。プロセス定義オブジェクトを作成すると，キューマネジャがその情報を取り出してトリガメッセージに設定します。この情報はトリガモニタアプリケーションから利用できます。キューに対応するプロセス定義オブジェクトの名前は，ローカルキュー属性の ProcessName 属性に指定します。各キューに異なるプロセス定義オブジェクトを指定したり，複数のキューで同じプロセス定義オブジェクトを共有したりできます。

トリガに関係する要素について説明します。

アプリケーションキュー

トリガが設定され，条件が満たされるとトリガメッセージの書き込みを要求するローカルキューです。

プロセス定義オブジェクト

アプリケーションキューには，プロセス定義オブジェクトを対応づけることができます。プロセス定義オブジェクトには，アプリケーションキューからメッセージを取り出すアプリケーションについての詳細情報があります。

トリガイベント

キューマネジャによってトリガメッセージが生成されるイベントです。通常は，アプリケーションキューへのメッセージの到着のことですが，その他の場合に発生することもあります。TP1/Message Queue では，トリガイベントを発生させる条件を制御できます。詳細については，「13.3　トリガイベントの条件」を参照してください。

トリガメッセージ

キューマネジャは，トリガイベントを検出するとトリガメッセージを生成します。このとき，キューマネジャは開始するアプリケーションについてのトリガメッセージ情報をコピーします。この情報はアプリケーションキュー，および対応するプロセス定義オブジェクトに依存します。トリガメッセージの形式については，「13.7.4　トリガメッセージの形式」を参照してください。

イニシエーションキュー

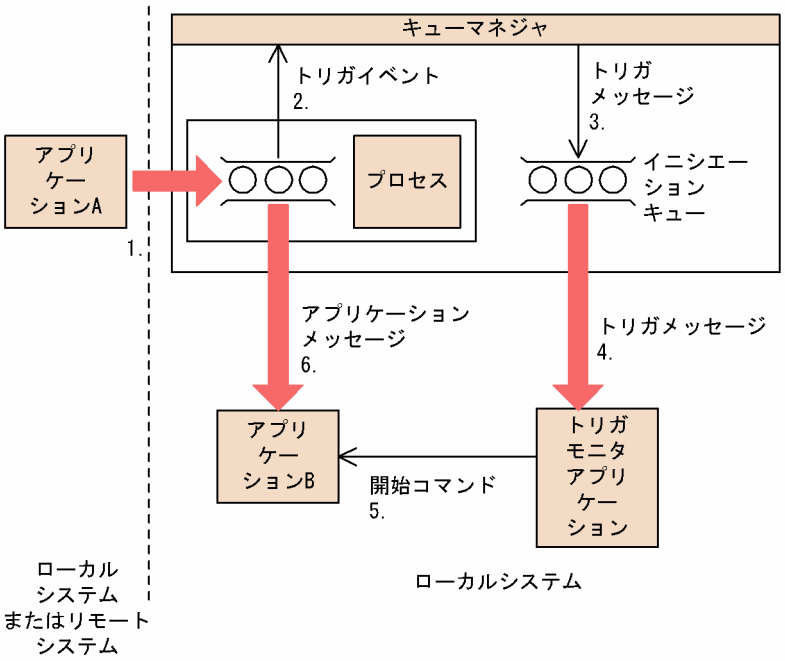
イニシエーションキューは，キューマネジャがトリガメッセージを登録するローカルキューです。キューマネジャは，一つ以上のイニシエーションキューを保持でき，各イニシエーションキューは一つ以上のアプリケーションキューと対応づけられます。

トリガモニタアプリケーション

トリガモニタアプリケーションは常時動作するプログラムであり，一つ以上のイニシエーションキューを処理します。トリガメッセージがイニシエーションキューに到着するとき，トリガモニタアプリケーションはトリガメッセージを取り出して情報を使用します。トリガモニタアプリケーションは他アプリケーションを開始させ，他アプリケーションはアプリケーションキューに到着したメッセージを取り出します。このときトリガモニタアプリケーションは，トリガメッセージヘッダにある情報を他アプリケーションに渡します。トリガメッセージヘッダには，アプリケーションキューの名前などがあります。

トリガの動作について，次の図に示します。この例では，トリガタイプは first です。

図 13–1　トリガの動作



図中の番号について説明します。

1. アプリケーション A はローカルシステムまたはリモートシステム上で動作し，アプリケーションキューにメッセージを登録します。取り出し用にこのキューをオープンするアプリケーションがないことに注意してください。このことはトリガタイプが first または depth である場合に影響します。
2. キューマネジャは，トリガイベントを生成する必要があるか条件を確認します。条件が満たされるとトリガイベントが生成されます。トリガメッセージの生成時には，対応するプロセス定義オブジェクトで保持する情報が使用されます。
3. キューマネジャはトリガメッセージを生成し，アプリケーションキューに対応するイニシエーションキューに登録します。ただし，トリガモニタアプリケーションが取り出し用にイニシエーションキューをオープンしているとします。

4. トリガモニタアプリケーションがイニシエーションキューからメッセージを取り出します。
5. トリガモニタアプリケーションが開始コマンドでアプリケーション B（サーバアプリケーション）を開始します。
6. アプリケーション B はアプリケーションキューをオープンして，メッセージを取り出します。

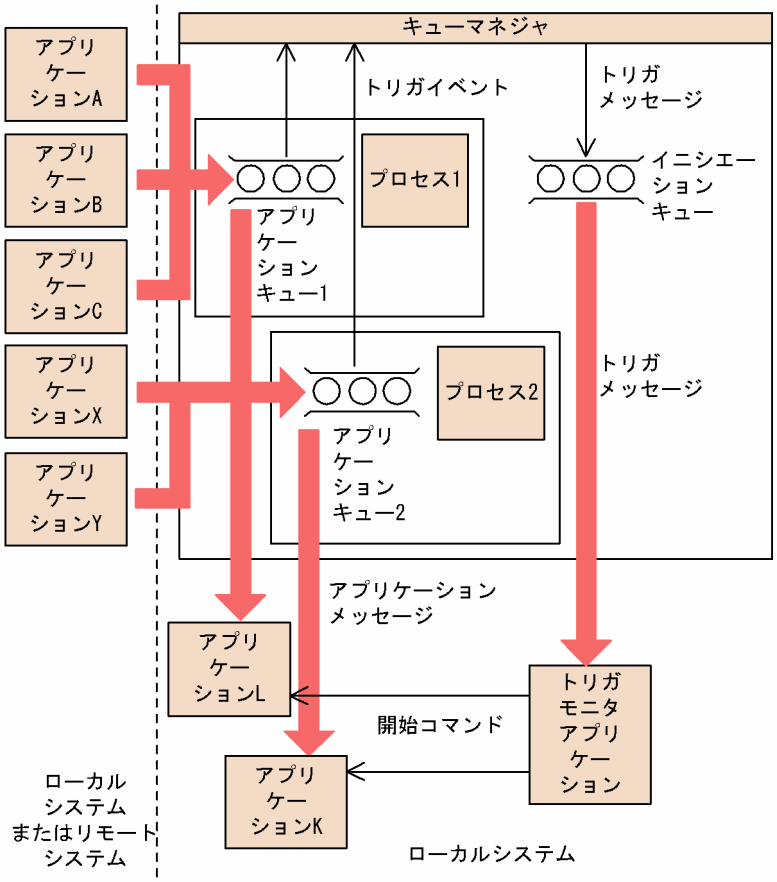
注意

1. 取り出し用にアプリケーションキューをオープンするアプリケーションがある場合にトリガタイプが first または depth であると，トリガイベントは発生しません。キューがすでに処理されているためです。
2. イニシエーションキューが取り出し用にオープンされていない場合，キューマネジャはトリガメッセージを生成しません。トリガモニタアプリケーションがイニシエーションキューを取り出し用にオープンするまで待ちます。
3. チャネルをトリガ起動する場合は，first または depth のトリガタイプにすることをお勧めします。

一つのイニシエーションキューで，複数のアプリケーションキューに対応できます。

アプリケーションキューとイニシエーションキューの関係について，次の図に示します。

図 13–2　アプリケーションキューとイニシエーションキューの関係



各アプリケーションキューには，対応するプロセス定義オブジェクトが一つあります。プロセス定義オブジェクトにはメッセージを処理するアプリケーションについての詳細情報を設定します。その情報をキュー

マネジャがトリガメッセージに設定するので，必要なイニシエーションキューは一つだけです。トリガモニタアプリケーションはトリガメッセージから情報を取り出し，各アプリケーションキューのメッセージを処理する適切なアプリケーションを開始します。

13.2.1　チャネルのトリガ起動

TP1/Message Queue ではローカルの転送キューにメッセージを登録することによってトリガイベントを発生させ，MQT サーバにチャネルを開始させることができます。これをトリガ起動といいます。詳細については，マニュアル「TP1/Message Queue 使用の手引」を参照してください。

13.3 トリガイベントの条件

次に示す条件が満たされるとき，キューマネジャはトリガメッセージを生成します。

1. メッセージがキューに登録されること。
2. メッセージの優先度が，トリガのためのメッセージ優先度の下限値以上であること。この下限値はローカルキューの TriggerMsgPriority 属性に指定します。0 を指定すると，すべてのメッセージでトリガメッセージが発生します。
3. TriggerMsgPriority 属性値以上の優先度のメッセージが，キューにすでにあること。メッセージ数は TriggerType 属性によって異なります。
   - 0（TriggerType 属性が MQTT_FIRST のとき）
   - 任意の数（TriggerType 属性が MQTT_EVERY のとき）
   - TriggerDepth 属性値－1（TriggerType 属性が MQTT_DEPTH のとき）

   注意

   キューマネジャは，トリガイベントの条件を満たすメッセージを数えるときに，コミット済みであるかコミット待ちであるかに関係なく数えます。そのため，キューのメッセージがコミット待ちであるために，キューにメッセージがない状態でアプリケーションが開始されることもあります。この場合は，アプリケーションがメッセージの到着を待ち合わせるように，WaitInterval フィールドを適切に指定して待ち合わせオプションを使用することをお勧めします。
4. トリガタイプが first または depth の場合は，メッセージの取り出し用にアプリケーションキューをオープンしているアプリケーションがないこと。つまり，ローカルキューの OpenInputCount 属性が 0 であること。
5. 次に示すどちらかの条件を満たすこと。
   - ローカルキューの ProcessName 属性が空白でなく，該当するプロセス定義オブジェクトが作成されていること。
   - ProcessName 属性が空白の転送キューであること。プロセス定義がオプションなので，開始するチャネル名が TriggerData 属性に含まれることがあります。この場合，トリガメッセージには次に示す値が設定されます。

     QName 属性：キュー名

     ProcessName 属性：空白

     TriggerData 属性：トリガデータ

     ApplType 属性：MQAT_UNKNOWN

     ApplId 属性：空白

     EnvData 属性：空白

     UserData 属性：空白
6. イニシエーションキューが作成され，InitiationQName 属性に指定されていること。イニシエーションキューは，次に示す設定にしてください。

- 取り出し操作を許可すること（InhibitGet 属性が MQQA_GET_ALLOWED）
- 登録操作を許可すること（InhibitPut 属性が MQQA_PUT_ALLOWED）
- Usage 属性を MQUS_NORMAL にすること
- 論理的に削除されている動的キューにしないこと

7. トリガモニタアプリケーションでイニシエーションキューをメッセージの取り出し用にオープンしていること。つまり，ローカルキューの OpenInputCount 属性が 0 より大きいこと。
8. アプリケーションキューのトリガ制御（TriggerControl 属性）が MQTC_ON であること。
9. トリガタイプ（TriggerType 属性）が MQTT_NONE でないこと。

   上記の条件 1 から 9 が満たされ，トリガ条件を発生させたメッセージがトランザクション内で登録される場合では，コミットとロールバックに関係なく，トランザクションが完了するまでトリガモニタアプリケーションからトリガメッセージは利用できません。トランザクションがコミットされるか，ロールバックされるか（トリガタイプが MQTT_FIRST または MQTT_DEPTH の場合）に関係なく利用できません。
10. 適切なメッセージがキューに登録される場合に TriggerType 属性が MQTT_FIRST または MQTT_DEPTH のキューが，それぞれ次に示す条件を満たすこと。
    - 事前に空でないこと（MQTT_FIRST のとき）
    - TriggerDepth 属性値以上のメッセージがあること（MQTT_DEPTH のとき）

    上記の条件 2 から 9（3 を除く）が満たされ，MQTT_FIRST のときは，最後のトリガメッセージがキューに登録されてから必要なトリガ間隔（キューマネジャ定義の TriggerInterval 属性）が経過していなければなりません。

    これによって，キューのすべてのメッセージを処理する前にキューサーバを終了できるようになります。トリガ間隔の目的は重複するトリガメッセージの数を減らすことです。

    注意

    キューマネジャを終了したあとで再度開始する場合は，トリガ間隔のタイマはリセットされます。
11. TriggerType 属性が MQTT_FIRST または MQTT_DEPTH のキューを処理する，単一のアプリケーションが MQCLOSE 命令を発行すること。そのとき少なくとも次に示す数のメッセージがキューにあること。
    - 1（MQTT_FIRST のとき）
    - TriggerDepth 属性値（MQTT_DEPTH のとき）

    キューのメッセージには必要な優先度（条件 2）があり，条件 5 から 9 も満たすようにしてください。

    これによって MQGET 命令を発行するキューサーバは，キューが空であることを検出して終了できるようになります。ただし，MQGET 命令と MQCLOSE 命令の間に一つ以上のメッセージが到着することがあります。

    注意

    a.アプリケーションキューを処理するアプリケーションですべてのメッセージを取り出さない場合，処理はループします。これは，アプリケーションがキューをクローズするたびに，キューマネジャ

が追加のトリガメッセージを生成し，トリガモニタアプリケーションにサーバプログラムを再度開始させるためです。

b.キューをクローズする前に，アプリケーションキューを処理するアプリケーションが取り出し要求をロールバックする場合，またはアプリケーションが中断する場合も同じループが発生します。

c.上記のループを防止するには，MQMD 構造体の BackoutCount フィールドを使用し，繰り返しロールバックされたメッセージを検出してください。詳細については，「3.10　ロールバックされたメッセージ」を参照してください。

12. MQSET 命令を使用して次に示す条件を満たすこと。

a.

- TriggerControl 属性を MQTC_ON に変更すること
- TriggerControl 属性がすでに MQTC_ON の場合に TriggerType，TriggerMsgPriority，または必要に応じて TriggerDepth 属性を変更すること

また，このとき少なくとも次に示す数のメッセージがキューにあること

- 1（MQTT_FIRST または MQTT_EVERY のとき）
- TriggerDepth 属性値（MQTT_DEPTH のとき）

キューのメッセージには必要な優先度（条件 2）があり，条件 4 から 9 も満たすようにしてください。

これによって，トリガ条件がすでに満たされているときにアプリケーションからトリガ基準を変更できます。

b.

イニシエーションキューの InhibitPut 属性を MQQA_PUT_INHIBITED から MQQA_PUT_ALLOWED に変更するときに，少なくとも次に示す数のメッセージがキューにあること

- 1（MQTT_FIRST または MQTT_EVERY のとき）
- TriggerDepth 属性値（MQTT_DEPTH のとき）

イニシエーションキューのメッセージには必要な優先度（条件 2）があり，条件 4 から 9 も満たすようにしてください。一つのトリガメッセージが条件を満たす各キューについて生成されます。

これによって，イニシエーションキューに MQQA_PUT_INHIBITED を設定してあとから変更するときにトリガメッセージを発生させることができます。

c.

イニシエーションキューの InhibitGet 属性が MQQA_GET_INHIBITED から MQQA_GET_ALLOWED に変更されるときに，少なくとも次に示す数のメッセージがキューにあること

- 1（MQTT_FIRST または MQTT_EVERY のとき）
- TriggerDepth 属性値（MQTT_DEPTH のとき）

イニシエーションキューのメッセージには必要な優先度（条件 2）があり，条件 4 から 9 も満たすようにしてください。

これによって，アプリケーションキューからメッセージを取り出せるときにだけアプリケーションをトリガするようにできます。

d.

トリガモニタアプリケーションがイニシエーションキューからの取り出し用に MQOPEN 命令を発行するときに，少なくとも次に示す数のメッセージがキューにあること

- 1（MQTT_FIRST または MQTT_EVERY のとき）
- TriggerDepth 属性値（MQTT_DEPTH のとき）

イニシエーションキューのメッセージには必要な優先度（条件 2）があり，条件 4 から 9 も満たすようにしてください。また，他アプリケーションが取り出し用にイニシエーションキューをオープンしていない状態にしてください。条件を満たす各キューについて一つのトリガメッセージが生成されます。

これによって，トリガモニタアプリケーションが動作していない間にキューに到着したメッセージ，およびキューマネジャの再度の開始で失ったトリガメッセージ（永続メッセージでない）についてトリガを発生させることができます。

注意

1. 条件 11 以降（アプリケーションキューへのメッセージ到着以外のイベントでのトリガメッセージ生成）では，トリガメッセージはトランザクションとして登録されません。また，TriggerType 属性が MQTT_EVERY の場合にアプリケーションキューに一つ以上のメッセージがあるときは，トリガメッセージが一つだけ生成されます。
2. MQPUT 命令の発行時にメッセージをセグメント分割する場合は，キューにすべてのセグメントが登録されるまでトリガイベントは処理されません。しかし，メッセージセグメントがいったんキューに登録されると，TP1/Message Queue は各セグメントを独立したトリガのためのメッセージとして処理します。例えば，3 分割された一つの論理メッセージでは，最初の MQPUT 命令でセグメント分割されたときに一つのトリガイベントが発生します。

13.4 トリガイベントの制御

アプリケーションキューを定義する属性を使用することによって，ユーザはトリガイベントを制御できます。トリガを有効にしたり無効にしたり，トリガイベント用に数えるメッセージの数や優先度を選択したりできます。

関連する属性について次に示します。

TriggerControl

アプリケーションキューのトリガを有効にしたり無効にしたりします。

TriggerMsgPriority

トリガイベント用に数えるメッセージの優先度の下限値です。トリガメッセージの生成の有無を決定するときにこの属性よりも低い優先度のメッセージがアプリケーションキューに到着した場合，キューマネジャはメッセージを無視します。0 を設定する場合は，すべてのメッセージがトリガイベント用に数えられます。

TriggerType

キューでトリガイベントを発生させる条件について，次に示すトリガタイプを指定できます。

none

TriggerControl 属性に off を設定した場合のようにトリガを無効にします。

every

アプリケーションキューにメッセージが到着するたびにトリガイベントを発生させます。1 メッセージを処理したあとでサーバアプリケーションを終了したい場合は，これを使用してください。

first

アプリケーションキューにあるメッセージの数が 0 から 1 に変わるときだけにトリガイベントを発生させます。キューに最初のメッセージが到着するときにサーバアプリケーションを開始させ，処理する必要があるメッセージがなくなるまで処理を継続したあとで終了したい場合は，これを使用してください。

depth

アプリケーションキューにあるメッセージの数が TriggerDepth 属性値に達したときにだけトリガイベントを発生させます。要求に対するすべての応答を受信したときにアプリケーションを開始したい場合は，これを使用してください。

注意

depth を指定する場合は，キューマネジャがトリガメッセージを生成したあとでトリガ（TriggerControl 属性）が無効になります。その場合は，アプリケーションで，MQSET 命令を使用してトリガを再度有効にしてください。

トリガを無効にする処理は，トランザクションに参加できません。そのため，トランザクションをロールバックするだけではトリガは再度有効にできません。トリガイベントを発生させた登録要求をアプリケーションでロールバックするときは，MQSET 命令でトリガを再度有効にしてください。

TriggerDepth

depth の指定時にトリガイベントを発生させるキューのメッセージ数です。

キューマネジャがトリガメッセージを生成する条件については，「13.3　トリガイベントの条件」を参照してください。

13.4.1　トリガタイプ every の使用例

保険の申し込みをするアプリケーションを例とします。アプリケーションは問い合わせメッセージを複数の保険会社に送信します。そのとき，同じ応答キューを指定します。応答が到着するたびに応答を処理するサーバのインスタンスが開始されるようにするには，応答キューに every のトリガタイプを指定します。

13.4.2　トリガタイプ first の使用例

複数の支店があって，1 日の取り引き明細を本店に送信する組織を例とします。週末に各支店から同時に送信します。本店ではアプリケーションが動作し，各支店からの取り引き明細を処理します。本店に最初に到着したメッセージがトリガイベントを発生させ，トリガイベントがアプリケーションを開始します。このアプリケーションはキューにメッセージがなくなるまで処理を継続します。

13.4.3　トリガタイプ depth の使用例

飛行機，ホテル，レンタルカー，およびその他チケットの予約を確認するために単一の問い合わせメッセージを生成する，旅行代理店アプリケーションを例とします。アプリケーションでは各項目を四つの問い合わせメッセージに分け，それぞれ別のあて先に送信します。応答キューには depth のトリガタイプ（TriggerDepth 属性は 4 に設定）を指定し，四つの応答すべてが到着したときにだけアプリケーションを再度開始するようにできます。

四つの応答の最後よりも先に他メッセージ（別の問い合わせに基づく）が応答キューに到着する場合は，問い合わせアプリケーションが早期に開始されます。これを回避するには，問い合わせについての複数の応答を格納するために depth を使用するときに，各問い合わせについて常に新規の応答キューを使用してください。

13.4.4　トリガタイプ first の特別な使用例

トリガタイプが first のアプリケーションキューに追加のメッセージが到着したときにすでにメッセージがある場合は，通常，キューマネジャは追加のトリガメッセージを生成しません。しかし，キューを処理するアプリケーションで（アプリケーションの終了やシステム障害などのため）キューをオープンできないこともあります。不正なアプリケーション名がプロセス定義オブジェクトに設定された場合，キューを処

理するアプリケーションはメッセージを取り出しません。この場合に追加のメッセージがアプリケーションキューに到着しても，メッセージを処理するサーバが動作していないことになります。

これに対処するには，該当するキューの最後のトリガメッセージを生成してから一定の時間が経過した場合にだけ，追加メッセージがアプリケーションキューに到着したときにキューマネジャで追加のトリガメッセージを生成するようにします。この時間はキューマネジャの TriggerInterval 属性に指定します。TP1/Message Queue の場合，デフォルトの値は 999999999 ミリ秒です。

アプリケーションで使用するためにトリガ間隔を指定するときには，次に示す項目について検討してください。

- TriggerInterval 属性に小さな値を設定した場合に，そのキューを処理するアプリケーションがないときは，first のトリガタイプが every のように動作することがあります。この動作はアプリケーションキューにメッセージが登録される頻度，つまり，相手システムの動作状態によって異なります。また，トリガ間隔に 0 を指定する first のトリガタイプは，every と同等です。
- トランザクションをロールバックする場合にトリガ間隔に大きな値（デフォルトを含む）を指定するときは，ロールバック時に一つのトリガメッセージが生成されます。しかし，トリガ間隔に小さな値または 0 を指定するとき（first のトリガタイプを every のように動作させる）は，複数のトリガメッセージが生成されます。トランザクションをロールバックしても，すべてのトリガメッセージは利用できる状態のままです。トリガメッセージの数は，トリガ間隔，およびトリガ間隔に 0 を指定するときの最大到着メッセージ数によって決定されます。

13.5　トリガ使用時のアプリケーションの設計

アプリケーション設計時に検討する必要がある項目について説明します。

13.5.1　トリガメッセージとトランザクション

トランザクション外のトリガイベントで生成されたトリガメッセージは，イニシエーションキューに登録され，トランザクションの外にあり，他メッセージとの依存関係はなく，トリガモニタアプリケーションによって取り出されるようになります。

トランザクション内のトリガイベントで生成されたトリガメッセージは，同じトランザクションの一部としてイニシエーションキューに登録されます。トリガモニタアプリケーションは，トランザクションが完了するまでこれらのトリガメッセージを取り出せません。トランザクションをコミットするときもロールバックするときも同様です。

キューマネジャがイニシエーションキューにトリガメッセージを登録するのに失敗する場合，トリガメッセージはデッドレターキューに登録されます。

注意

トリガイベントの条件があることを確認するためにアクセスするとき，キューマネジャはコミットされたメッセージとコミット待ちメッセージの両方を数えます。

トリガタイプが first または depth の場合は，トランザクションがロールバックされるときにも，必要な条件が満たされるときにトリガメッセージを利用できます。例えば，first のトリガタイプのキューにトランザクション内で登録を実行するとします。この場合，キューマネジャはトリガメッセージを生成します。追加の登録がほかのトランザクションで発生しても，追加のトリガイベントは発生しません。アプリケーションキューのメッセージ数が 1 から 2 に変更されるために，トリガイベントの条件が満たされないからです。最初のトランザクションがロールバックされ，次のトランザクションがコミットされると，トリガメッセージはまた生成されます。

しかし，このことによって，トリガイベントの条件が満たされないときにもトリガメッセージが生成されることがあります。トリガを使用するアプリケーションは常に状況に対応できるようにしてください。MQGET 命令に待ち合わせオプションを使用し，WaitInterval フィールドに適切な値を設定することをお勧めします。

13.5.2　トリガが設定されたキューからのメッセージ取り出し

トリガを使用するアプリケーションを設計するときは，トリガモニタアプリケーションによってアプリケーションが開始されてから，追加メッセージがアプリケーションキューで利用できるようになるまでに遅延があることに注意してください。このことは，トリガイベントを発生させたメッセージがほかのメッセージよりも先にコミットされる場合に発生します。

メッセージが到着する時間を待つには，トリガ条件が設定されたキューから MQGET 命令を使用してメッセージを取り出すときに待ち合わせオプションを設定します。メッセージが登録されてから登録命令がコミットされるまでの時間よりも，十分に余裕のある時間を WaitInterval フィールドに設定してください。リモートキューマネジャからメッセージが到着する場合には，次に示す項目の影響を受けます。

- コミット前に登録されたメッセージの数
- 回線の速度と可用性
- メッセージの長さ

待ち合わせオプションを指定する MQGET 命令の使用方法については，トランザクションを記述する場合を参考にしてください。ここでは，トリガタイプ first のキューにトランザクション内でメッセージが登録されたとします。このイベントでキューマネジャはトリガメッセージを生成します。追加の登録がほかのトランザクション内で発生しても，追加のトリガイベントは発生しません。これは，アプリケーションキューのメッセージの数が 0 から 1 になるわけではないためです。最初のトランザクションがロールバックされ，第 2 のトランザクションがコミットされると，トリガメッセージはまた生成されます。つまり，最初のトランザクションがロールバックされるときにトリガメッセージが生成されます。第 2 のメッセージがコミットされるまでに遅延がある場合には，アプリケーションで待ち合わせるようにしてください。

depth のトリガタイプでは，関連するすべてのメッセージを最後にコミットするときでも遅延が発生します。TriggerDepth 属性に 2 が設定されているとします。二つのメッセージがキューに到着するとき，二つ目のメッセージでトリガメッセージが生成されます。そして，二つ目のメッセージが最初にコミットされるときに，トリガメッセージが利用できるようになります。トリガモニタアプリケーションは，サーバアプリケーションを開始しますが，最初のメッセージがコミットされるまでは第 2 のメッセージだけしかアプリケーションで取り出しできません。そのため，アプリケーションは最初のメッセージが利用できるようになるまで待ち合わせなければなりません。

待ち合わせ時間が経過したときに取り出せるメッセージがない場合は，終了するようにアプリケーションを設計してください。一つ以上のメッセージが連続して到着する場合には，処理するアプリケーションを再度開始するようにしてください。アプリケーションをむだに動作させることで不要なリソースが消費されるのを防止できます。

13.6　トリガモニタアプリケーション

キューマネジャにとっては，トリガモニタアプリケーションはキューを処理する他アプリケーションと同等です。ただし，トリガモニタアプリケーションはイニシエーションキューを処理します。

トリガモニタアプリケーションは，通常は常時動作するアプリケーションです。トリガメッセージがイニシエーションキューに到着するとき，トリガモニタアプリケーションはメッセージを取り出します。そして，メッセージ内の情報を使用して，アプリケーションキューのメッセージを処理するアプリケーションを開始するコマンドを発行します。

アプリケーションが適切なアプリケーションキューに対して適切な動作を実行できるように，トリガモニタアプリケーションから十分な情報をアプリケーションに渡してください。

13.7　トリガメッセージの属性

トリガメッセージのその他の属性について説明します。

13.7.1　トリガメッセージの永続性と優先度

特に指定しない場合，トリガメッセージは永続メッセージになりません。しかし，トリガイベントが発生する条件は永続的なので，条件が満たされるときにはいつでもトリガメッセージが生成されます。トリガメッセージを失った場合，アプリケーションキューにアプリケーションメッセージが残っていれば，条件が満たされるときにキューマネジャはトリガメッセージを生成します。

トランザクションがロールバックされる場合，生成されたトリガメッセージは常に転送されます。

トリガメッセージはイニシエーションキューのデフォルトの優先度を取得します。

13.7.2　キューマネジャの再度開始とトリガメッセージ

キューマネジャを再度開始したあとでイニシエーションキューを取り出し用にオープンするとき，対応するアプリケーションキューにメッセージがあってトリガが設定されていると，イニシエーションキューにトリガメッセージが登録されます。

13.7.3　トリガメッセージとオブジェクト属性の変更

トリガメッセージは有効なトリガ属性の値に従って生成されます。メッセージがトランザクション内で登録されるなどして，トリガモニタアプリケーションでトリガメッセージを利用できるのがあとになる場合は，トリガ属性の変更はしばらくトリガメッセージに影響しません。また，トリガを無効にする場合もいったん生成されたトリガメッセージは利用できないようにできません。トリガメッセージが利用できるようになった時点で，アプリケーションキューがないこともあります。

13.7.4　トリガメッセージの形式

トリガメッセージの形式は，MQTM 構造体で定義されます。この構造体には，次に示すフィールドがあります。キューマネジャはトリガメッセージを生成するとき，アプリケーションキューのオブジェクト定義およびプロセス定義を使用して，構造体に情報を設定します。

StrucId

　構造体識別子

Version

構造体バージョン

QName

トリガイベントが発生したアプリケーションキューの名前。キューマネジャがトリガメッセージを生成するとき，このフィールドにアプリケーションキューの QName 属性を設定します。

ProcessName

アプリケーションキューに対応するプロセス定義オブジェクトの名前。キューマネジャがトリガメッセージを生成するとき，このフィールドにアプリケーションキューの ProcessName 属性を設定します。

TriggerData

トリガモニタアプリケーションで使用するための自由形式の領域。キューマネジャがトリガメッセージを生成するとき，このフィールドにアプリケーションキューの TriggerData 属性を設定します。

ApplType

トリガモニタアプリケーションが開始するアプリケーションタイプ。キューマネジャがトリガメッセージを生成するとき，このフィールドに ProcessName 属性で識別されるプロセス定義オブジェクトの ApplType 属性を設定します。

ApplId

トリガモニタアプリケーションが開始するアプリケーションを識別する文字列。キューマネジャがトリガメッセージを生成するとき，このフィールドに ProcessName 属性で識別されるプロセス定義オブジェクトの ApplId 属性を設定します。

EnvData

トリガモニタアプリケーションで使用する環境関連データの文字フィールド。キューマネジャがトリガメッセージを生成するとき，このフィールドに ProcessName 属性で識別されるプロセス定義オブジェクトの EnvData 属性を設定します。

UserData

トリガモニタアプリケーションで使用するユーザデータの文字フィールド。キューマネジャがトリガメッセージを生成するとき，このフィールドに ProcessName 属性で識別されるプロセス定義オブジェクトの UserData 属性を設定します。

MQTM 構造体の詳細については，マニュアル「TP1/Message Queue プログラム作成リファレンス」を参照してください。

13.8　トリガの動作失敗

トリガモニタアプリケーションが開始できない場合，およびキューマネジャがトリガメッセージを転送できない場合には，アプリケーションは開始されません。

キューが満杯であったり，イニシエーションキューに登録可能な最大長よりもメッセージが長かったりすることなどが原因で，トリガメッセージが生成されてもイニシエーションキューに登録できなかった場合には，トリガメッセージは代わりにデッドレターキューに登録されます。

デッドレターキューへの登録に失敗した場合にはトリガメッセージは破棄され，TP1/Message Queue では，KFCA04259-E メッセージが出力されます。

デッドレターキューへのトリガメッセージの登録によって，該当するキューについてのトリガメッセージを生成できます。このトリガメッセージは，キューマネジャがデッドレターキューにメッセージを追加するときに破棄されます。

付録

付録 A　用語解説

MQI に関する用語について説明します。

(英字)

CCSID（coded character set identifier)
: 文字セット識別子を参照してください。

FIFO（first-in-first-out)
: キューイング手法の一つです。次に取り出す項目がキューの中で最も長く存在する項目になります。

IBM MQ
: 米国 IBM 社がライセンスするプログラム製品の一つです。メッセージキューイングサービスを提供します。

MCA（message channel agent)
: メッセージチャネルエージェントを参照してください。

MQI（message queue interface)
: メッセージキューインタフェースを参照してください。

PCF（programmable command format)
: プログラマブルコマンドフォーマットを参照してください。

(ア行)

アプリケーションキュー
: アプリケーションが使用するキューのことです。

一時的動的キュー
: クローズ時に削除される動的キューのことです。一時的動的キューはキューマネジャの失敗時に回復されません。そのため，非永続メッセージだけを保持します。永続的動的キューと比較してください。

イニシエーションキュー
: キューマネジャがトリガメッセージを登録するローカルキューです。

イベント

チャネルイベント，パフォーマンスイベント，およびキューマネジャイベントを参照してください。

イベントキュー

キューマネジャがイベントを検出したあとでイベントメッセージを登録するキューのことです。イベントの種類（チャネルイベント，パフォーマンスイベント，およびキューマネジャイベント）ごとに独自のイベントキューがあります。

イベントメッセージ

イベントの種類，イベント発生元のアプリケーション名，キューマネジャ統計などの情報があるメッセージのことです。

永続的動的キュー

削除を明示的に指示したときだけ，クローズ時に削除される動的キューのことです。キューマネジャの失敗時に永続的動的キューは回復されます。そのため，永続メッセージを保持できます。一時的動的キューと比較してください。

永続メッセージ

キューマネジャの再度の開始時に残るメッセージのことです。非永続メッセージと比較してください。

応答キュー

MQPUT 命令を発行したアプリケーションが応答メッセージまたは報告メッセージを受け取るために使用する，キューの名前のことです。

応答メッセージ

問い合わせメッセージに対する応答で使用するメッセージタイプのことです。問い合わせメッセージ，および報告メッセージを参照してください。

オブジェクト

TP1/Message Queue では，キューマネジャ，キュー，プロセス定義，またはチャネルのことです。

オブジェクト記述子

オブジェクトを識別するデータ構造体のことです。オブジェクト記述子はオブジェクトとオブジェクトタイプの名前を持っています。

オブジェクトハンドル

アプリケーションで操作したいオブジェクトにアクセスするための，識別子のことです。

（カ行）

解決されるキュー

MQOPEN 命令の入力として，アプリケーションが別名キューやリモートキューを指定するときにオープンされるキューのことです。

完了コード

MQI 命令の終了状態を表すリターンコードのことです。

キュー

オブジェクトの一つです。メッセージキューイングアプリケーションは，メッセージを登録したり，取り出したりできます。キューはキューマネジャによって保持されます。ローカルキューには，処理を待つメッセージが格納されます。他タイプのキューには，メッセージは格納されません。他キューを指し示したり，動的キューを作成したりするときのひな形として使用されます。

キューイング

メッセージキューイングを参照してください。

キューマネジャ

1. アプリケーションにキューイングサービスを提供するシステムプログラムのことです。キューマネジャが保持するキューにアプリケーションからアクセスするための，アプリケーションプログラミングインタフェースを提供します。ローカルキューマネジャ，およびリモートキューマネジャを参照してください。
2. オブジェクトの一つです。特定のキューマネジャの属性を定義します。

キューマネジャイベント

次に示す状況を表すイベントのことです。

- キューマネジャによって使用されるリソースに関連するエラー状態の発生（例：キューマネジャ使用不可）
- キューマネジャでの大きな変更の発生（例：キューマネジャの開始や停止）

組み込みフォーマット

メッセージキューイングの場合，キューマネジャで定義された意味のある，メッセージ内のアプリケーションデータのことです。ユーザ定義フォーマットと比較してください。

クラスタ

論理的に関連づけられたキューマネジャのネットワークのことです。

検索

メッセージキューイングの場合，MQGET 命令を使用してメッセージをキューから削除しないでコピーすることです。取り出しと比較してください。

検索カーソル

メッセージキューイングの場合，キューを検索するときに使用する指示子であり，次メッセージを示します。

コネクションハンドル

接続したキューマネジャにアプリケーションからアクセスするための識別子のことです。

コミット

トランザクションでのすべての操作を確定する操作のことです。操作が完了したあとで新規のトランザクションを開始できます。ロールバックと比較してください。

コンテキスト

メッセージ登録元についての情報のことです。

（サ行）

出力用パラメタ

命令の完了または失敗時に，キューマネジャが情報を返す MQI 命令のパラメタのことです。

接続

キューマネジャのコネクションハンドルを取得することです。アプリケーションで以降の MQI 命令を発行するときに使用します。MQCONN 命令で作成したり，MQOPEN 命令発行時に自動的に作成したりします。

属性

オブジェクトの性質を定義するプロパティのことです。

（タ行）

単相コミット

キューへの変更を，他リソースマネジャによって制御されるリソースへの変更と連携することなく，アプリケーションでコミットする方法のことです。

二相コミットと比較してください。

チャネル

メッセージチャネルを参照してください。

チャネルイベント

チャネルのインスタンスが利用できるようになったかどうかを表すイベントのことです。チャネルイベントは，チャネルの両端にあるキューマネジャ上で発生します。

デッドレターキュー

あて先への転送に失敗したメッセージをキューマネジャやアプリケーションが登録するためのキューのことです。

転送キュー

リモートキューマネジャあてのメッセージが一時的に格納されるローカルキューのことです。

問い合わせメッセージ

他アプリケーションからの応答を要求するメッセージで使用するメッセージタイプのことです。応答メッセージ，および報告メッセージを参照してください。

同期点

処理の中間や最後に位置し，保護リソースの一貫性が確定するポイントのことです。同期点では，リソースへの変更がそのままコミットされたり，ロールバックされて直前の同期点の状態に戻されたりします。

同期メッセージング

アプリケーションがメッセージをキューに登録する，プログラム間通信の 1 形態です。

同期メッセージングでは，送信側アプリケーションは自メッセージに対する応答を待ってから処理を進めます。非同期メッセージングと比較してください。

動的キュー

アプリケーションがモデルキューオブジェクトをオープンするときに作成されるローカルキューのことです。永続的動的キュー，および一時的動的キューを参照してください。

トランザクション

同期点間の区間で，アプリケーションから実行する回復可能な操作のことです。

TP1/Message Queue は，OpenTP1 システムのトランザクション制御に従います。

トリガイベント

キューへのメッセージ到着など，イニシエーションキューのトリガメッセージをキューマネジャに生成させるイベントのことです。

トリガ機能

キューに設定した条件が満たされるときに，キューマネジャに自動的にアプリケーションを開始させる機能のことです。

トリガメッセージ

トリガモニタアプリケーションが開始するための，アプリケーション情報を含むメッセージのことです。

トリガモニタアプリケーション

一つ以上のイニシエーションキューを処理する常時動作のアプリケーションのことです。トリガメッセージがイニシエーションキューに到着するとき，トリガモニタアプリケーションはメッセージを取り出します。トリガモニタアプリケーションはトリガメッセージ内の情報を使用して，トリガイベント発生元のキューを処理するアプリケーションを開始します。

取り出し

メッセージキューイングの場合，MQGET 命令を発行してメッセージをキューから削除することです。検索と比較してください。

（ナ行）

二相コミット

一つのトランザクションで複数のリソースマネジャが使用されるときに，回復可能なリソースの変更を連携させる方法のことです。

単相コミットと比較してください。

入出力用パラメタ

命令発行時にユーザが情報を設定し，命令の完了または失敗時にキューマネジャが情報を変更する MQI 命令のパラメタのことです。

入力用パラメタ

命令発行時にユーザが情報を設定するパラメタのことです。

認証確認

ユーザがオブジェクトに対して命令を発行するときに実行されるセキュリティ確認のことです。例えば，キューをオープンするときやキューマネジャに接続するときに実行されます。

（ハ行）

配布リスト

単一の MQPUT 命令または MQPUT1 命令で，メッセージを登録したいキューの一覧のことです。

パフォーマンスイベント

制限状態の発生を示すイベントのことです。

ハンドル

コネクションハンドルおよびオブジェクトハンドルを参照してください。

非永続メッセージ

キューマネジャの再度の開始時には残らないメッセージのことです。永続メッセージと比較してください。

非問い合わせ

TP1/Message Queue がサポートする最も単純なメッセージのことです。応答を要求しません。

非同期メッセージング

アプリケーションがメッセージをキューに登録する，プログラム間通信の 1 形態です。

非同期メッセージングでは，送信側アプリケーションは自メッセージに対する応答を待たないで処理を進めます。同期メッセージングと比較してください。

フォーマット

メッセージキューイングの場合，メッセージ中のアプリケーションデータの性質を表す用語です。組み込みフォーマットおよびユーザ定義フォーマットを参照してください。

プログラマブルコマンドフォーマット

メッセージタイプの一つです。次に示すプログラムで使用します。

- 指定したキューマネジャのシステムコマンド入力キューに PCF コマンドを登録する，ユーザ管理アプリケーション
- 指定したキューマネジャから PCF コマンドの結果を取得する，ユーザ管理アプリケーション
- イベントの発生を通知するキューマネジャ

プロセス定義オブジェクト

アプリケーションの定義を保持するオブジェクトのことです。例えば，キューマネジャがトリガメッセージを生成するときに使用します。

分散アプリケーション

メッセージキューイングの場合，それぞれが異なるキューマネジャに接続されながら全体で一つのアプリケーションを構成する，アプリケーションの集合のことです。

別名キューオブジェクト

このオブジェクトの名前は，ローカルキューマネジャで定義されるベースキューの別名です。アプリケーションまたはキューマネジャが別名キューを使用するときに別名は解決され，該当するベースキューに対して要求操作が実行されます。

報告メッセージ

他メッセージについての情報を持つメッセージタイプのことです。メッセージが転送されたこと，あて先に届いたこと，保持時間を超過したこと，または何かの理由で処理されなかったことを表します。応答メッセージ，および問い合わせメッセージと比較してください。

（マ行）

メッセージ

メッセージキューイングアプリケーションの場合，アプリケーション間で送信する通信文のことです。永続メッセージ，および非永続メッセージを参照してください。

メッセージ記述子

メッセージの一部として転送される制御情報のことです。メッセージフォーマットと表現形式を記述します。メッセージ記述子の形式は MQMD 構造体で定義されます。

メッセージキュー

キューを参照してください。

メッセージキューイング

プログラミング手法の一つです。各アプリケーションはメッセージをキューに登録することによって，相互に通信します。

メッセージキューインタフェース

キューマネジャが提供するプログラミングインタフェースのことです。これを使用してアプリケーションは，キューイングサービスにアクセスできます。

メッセージグループ

論理メッセージのグループのことです。メッセージを論理的にグループ化すると，アプリケーションで類似するメッセージをまとめたり，メッセージの論理的順序を確保したりできます。

メッセージセグメント

アプリケーションまたはキューマネジャで処理するには長過ぎるメッセージを分割した，セグメントの一つのことです。

メッセージチャネル

分散メッセージキューイングの場合，キューマネジャ間でメッセージを移動させる仕組みのことです。

メッセージチャネルエージェント

用意されたメッセージを転送キューから通信リンクに，または通信リンクからあて先キューに送信するプログラムのことです。

メッセージ優先度

メッセージ属性の一つです。キューのメッセージが取り出される順序やトリガイベント生成の可否に影響します。

メッセージング

同期メッセージング，および非同期メッセージングを参照してください。

文字セット識別子

文字コードの体系に割り当てられた名前のことです。

モデルキューオブジェクト

アプリケーションが動的キューを作成するときにひな形として使用する，キュー属性の組み合わせのことです。

（ヤ行）

ユーザ定義フォーマット

メッセージキューイングの場合，ユーザアプリケーションで定義された意味のある，メッセージ内のアプリケーションデータのことです。組み込みフォーマットと比較してください。

（ラ行）

リソースマネジャ

メモリバッファやデータセットなど共有リソースへのアクセスを管理および制御するための，アプリケーション，プログラム，または処理のことです。

IBM MQ や TP1/Message Queue はリソースマネジャです。

リターンコード

完了コードと理由コードの総称のことです。

リモートキュー

リモートキューマネジャに属するキューのことです。アプリケーションはリモートキューにメッセージを登録できますが，リモートキューからメッセージを取り出すことはできません。ローカルキューと比較してください。

リモートキューイング

メッセージキューイングの場合，アプリケーションにサービスを提供して，他キューマネジャに属するキューにメッセージを登録させることです。

リモートキューオブジェクト

リモートキューのローカル定義を参照してください。

リモートキューのローカル定義

ローカルキューマネジャに属するオブジェクトの一つです。他キューマネジャによって保持されるキューの属性を定義します。また，キューマネジャの別名や応答キューの別名にも使用します。

リモートキューマネジャ

自アプリケーションが接続しないキューマネジャのことです。

理由コード

MQI 命令の失敗や部分的完了の理由を表すリターンコードのことです。

類似性

オブジェクト間の関連や依存のことです。

ローカルキュー

ローカルキューマネジャに属するキューのことです。処理を待つメッセージが格納されます。リモートキューと比較してください。

ローカルキューマネジャ

アプリケーションから接続してメッセージキューイングサービスを受けるためのキューマネジャのことです。自アプリケーションを接続しないキューマネジャは，同じシステムで動作していても，リモートキューマネジャといいます。

ロールバック

トランザクションでのすべての変更を元に戻す操作のことです。操作が完了したあとで新規のトランザクションを開始できます。コミットと比較してください。

索引

か

き

く

け

こ

さ

し

す

せ

る

ろ