OpenTP1 Version 7

メッセージキューイングアクセス機能

TP1/Message Queue - Access
使用の手引

解説・手引・文法・操作書

3000-3-D94-10

マニュアルの購入方法

このマニュアル，および関連するマニュアルをご購入の際は，巻末の「ソフトウェアマニュアルのサービス ご案内」をご参照ください。

HITACHI

■対象製品

P-1M64-C771 uCosminexus TP1/Message Queue - Access 07-01（適用 OS：AIX 5L V5.1，AIX 5L V5.2，AIX 5L V5.3）
P-1J64-C871 uCosminexus TP1/Message Queue - Access（64）07-01（適用 OS：HP-UX 11i V2（IPF））※
P-9S64-C771 uCosminexus TP1/Message Queue - Access 07-01（適用 OS：Red Hat Enterprise Linux AS 4，Red Hat Enterprise Linux ES 4（x86，AMD64，Intel EM64T））※
P-9V64-C771 uCosminexus TP1/Message Queue - Access（64）07-01（適用 OS：Red Hat Enterprise Linux AS 4（IPF））※
P-2464-C774 uCosminexus TP1/Message Queue - Access 07-01（適用 OS：Windows Server 2003 x64 Editions，Windows Server 2003，Windows XP，Windows Vista（32 ビット用））※
これらのプログラムプロダクトのほかにも，このマニュアルをご利用になれる場合があります。詳細は「リリースノート」でご確認ください。
これらの製品は，ISO9001 および TickIT の認証を受けた品質マネジメントシステムで開発されました。
※印のプログラムプロダクトについては，発行時期をご確認ください。

■輸出時の注意

本製品を輸出される場合には，外国為替および外国貿易法ならびに米国の輸出管理関連法規などの規制をご確認の上，必要な手続きをお取りください。
なお，ご不明な場合は，弊社担当営業にお問い合わせください。

■商標類

■発行

2006 年 11 月 （第 1 版） 3000-3-D94
2007 年 9 月 （第 2 版） 3000-3-D94-10

変更内容

変更内容 (3000-3-D94-10)　uCosminexus TP1/Message Queue - Access　07-01

追加・変更内容	変更個所
適用 OS から Solaris を削除した。	2.1，表 2-8，表 2-9，表 2-10，表 2-11，mqcapiout
適用 OS から HP-UX 11i V2（PA-RISC）を削除した。	2.1
JMS インタフェースを使用する場合の環境変数の設定方法を変更した。	2.2.1，2.2.1(4)，2.2.1(4)(a)
非 XA 連携のアプリケーションで，複数のスレッドが異なるキューマネジャに接続できる機能を追加した。また，これに伴い，接続先情報定義ファイルに関連する環境変数のオペランドを追加した。	2.2.1(1)，2.2.1(2)，2.2.1(3)，2.2.1(6)，2.2.2(3)，2.2.5，表 A-1
TP1/Server Base と連携する場合，システム環境定義の $DCCONFPATH/env への設定形式を変更した。	2.2.1(5)
環境変数のオペランド DCMQCSRVHOSTNAME の指定できるホスト名の長さを変更した。	2.2.2(1)
環境変数のオペランド DCMQCTIMEGET の指定できる値の範囲および初期値を変更した。	2.2.2(1)
JMSPRF トレースファイルのパスを変更した。	表 2-12
MDB のキュー監視スレッドからの JMSPRF トレースファイルの情報取得機能を追加した。	2.4.4
mqcapiout（API トレースファイル取得）コマンドに，コネクションハンドルの情報を出力するオプション，および出力形式を追加した。	mqcapiout
メッセージグループの最終を設定できるプロパティを追加した。	表 7-6
メッセージセレクタ機能を追加した。また，使用できるようになったことに伴い，注意事項を追加，削除し，例外（InvalidSelectorException）を追加した。	メッセージセレクタ，MessageConsumer インタフェース，QueueBrowser インタフェース（getMessageSelector），QueueSession インタフェース（createBrowser），QueueSession インタフェース（createReceiver），表 B-2
public boolean hasMoreElements メソッド，および public Object nextElement の説明を変更した。	Enumeration インタフェース
メッセージを追加した。 KFCA30952-E ～ KFCA30954-W，KFCA30981-E，KFCA30982-E	8.2
メッセージを変更した。 KFCA30960-E，KFCA30961-E，KFCA30963-E，KFCA30964-E，KFCA30975-I，KFCA31341-W	8.2

単なる誤字・脱字などはお断りなく訂正しました。

はじめに

このマニュアルは，TP1/Message Queue - Access の機能，操作，および運用について説明したものです。
TP1/Message Queue - Access 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，WebSphere MQ の MQI の仕様をベースに実装しています。

■対象読者

TP1/Message Queue - Access を使用するシステム管理者およびシステム設計者で，OpenTP1 システムの知識がある方を対象としています。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。

- OpenTP1 解説（3000-3-D50）
- OpenTP1 プログラム作成の手引（3000-3-D51）
- OpenTP1 システム定義（3000-3-D52）
- OpenTP1 運用と操作（3000-3-D53）
- TP1/Message Queue 使用の手引（3000-3-D90）
- TP1/Message Queue プログラム作成の手引（3000-3-D92）
- TP1/Message Queue プログラム作成リファレンス（3000-3-D93）

■マニュアルの構成

このマニュアルは，次に示す章と付録から構成されています。

第 1 章　概要

TP1/Message Queue - Access（MQC）の接続形態や，MQC クライアント機能と MQC サーバ機能の関係などについて説明しています。

第 2 章　MQC クライアント機能

MQC クライアント機能の定義，クライアントアプリケーションの作成，および障害対策について説明しています。

第 3 章　MQC クライアント機能の運用コマンド

MQC クライアント機能の運用コマンドについて説明しています。

第 4 章　MQC クライアント機能の MQI

C 言語または COBOL 言語の MQI を使用したクライアントアプリケーションの作成方法について説明しています。

第 5 章　MQC クライアント機能の C++ インタフェース

MQC クライアント機能の C++ インタフェースについて説明しています。

第 6 章　MQC クライアント機能の Java インタフェース

MQC クライアント機能の Java インタフェースについて説明しています。

第 7 章　MQC クライアント機能の JMS インタフェース

MQC クライアント機能の JMS インタフェースについて説明しています。

第 8 章　メッセージの一覧

MQC クライアント機能が出力するメッセージについて説明しています。

付録 A　理由コード

MQC の理由コードについて説明しています。

付録 B　JMS 仕様と MQC クライアント機能の JMS インタフェースとの差異

Sun Microsystems, Inc. が提供する JMS 1.0 と MQC クライアント機能の JMS インタフェースとの機能差について説明しています。

付録 C　MQMessage クラスで変換できるコード化文字セット識別子の一覧

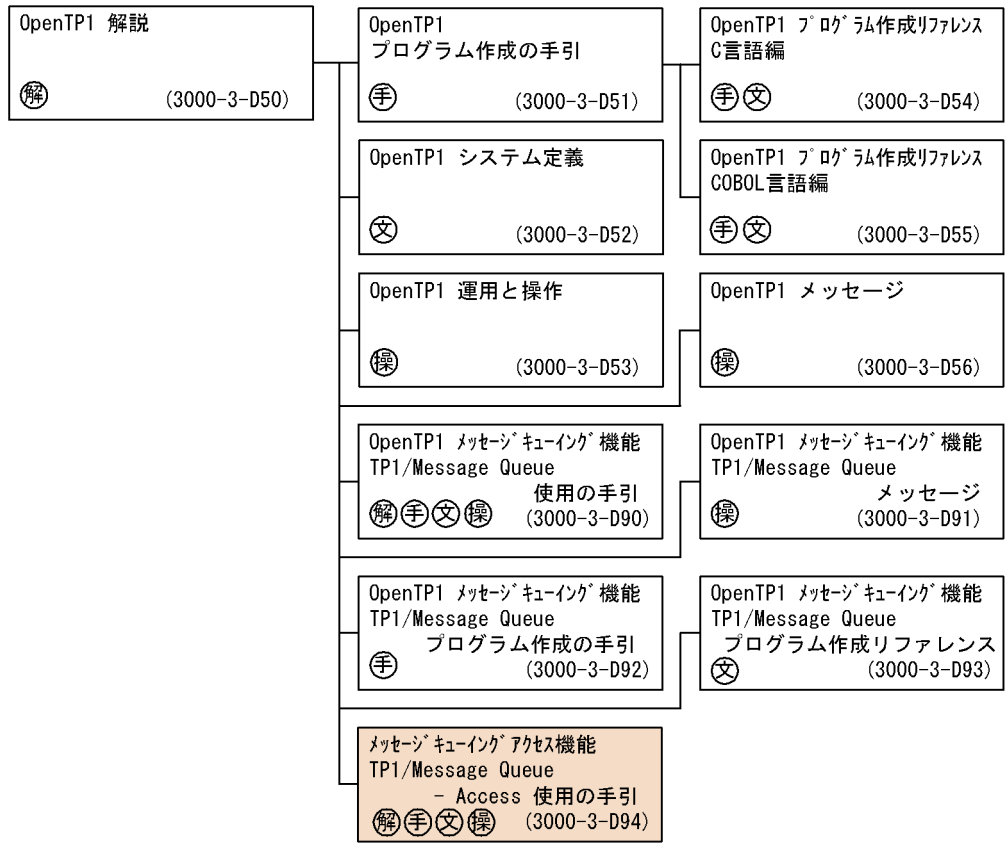
MQMessage クラスで変換できるコード化文字セット識別子について説明しています。

付録 D　用語解説

関係する用語について説明しています。

■関連マニュアル

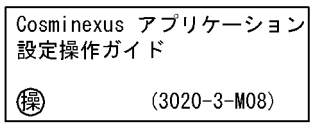
●OpenTP1 Version 7



●その他の関連製品

OpenTP1

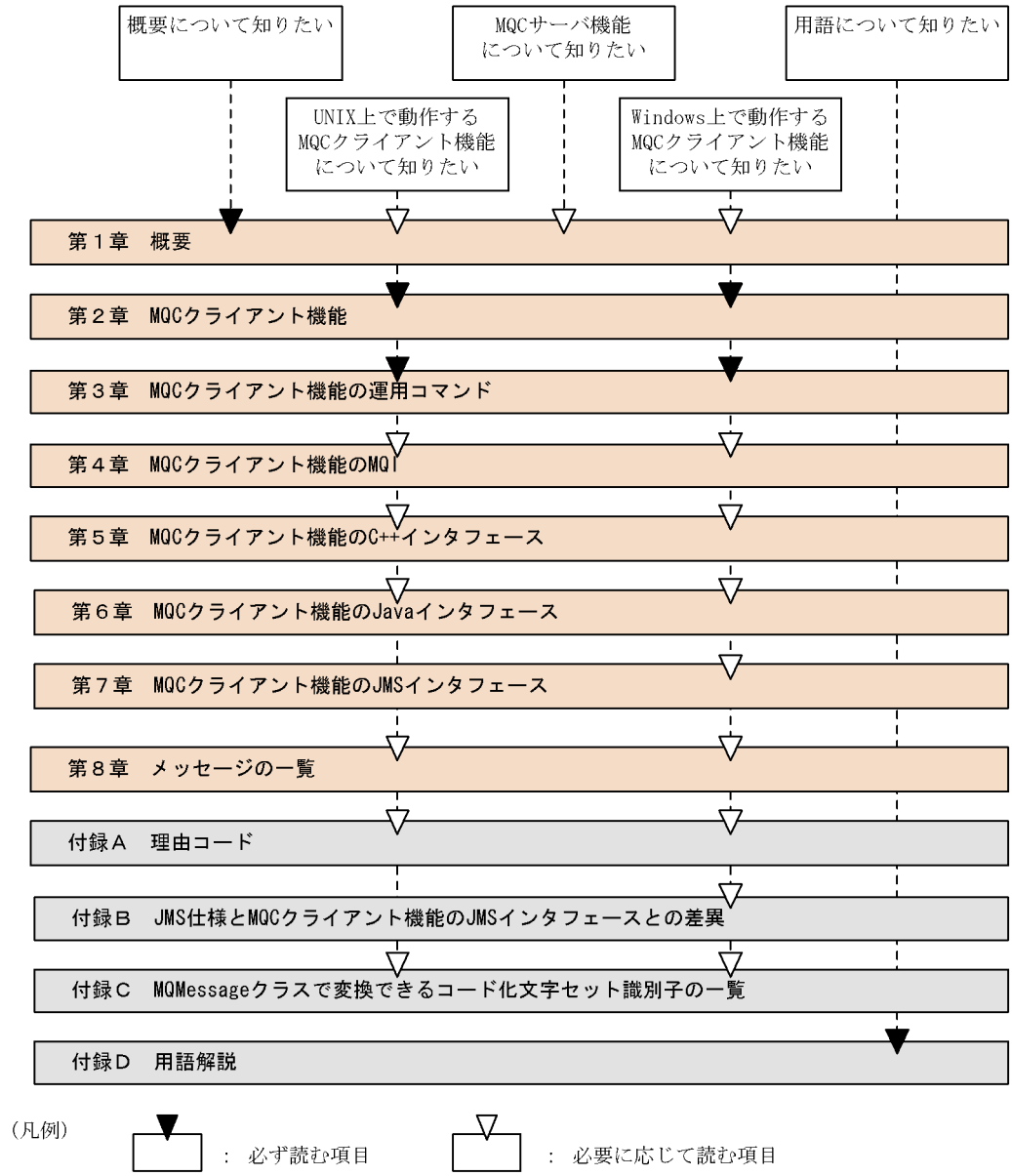
Cosminexus



〈記号〉
解 ：解説書
手 ：手引書
文 ：文法書
操 ：操作書

■読書手順

このマニュアルは，利用目的に合わせて直接章を選択して読むことができます。



■このマニュアルでの表記

このマニュアルで使用する製品名称の略称を次に示します。

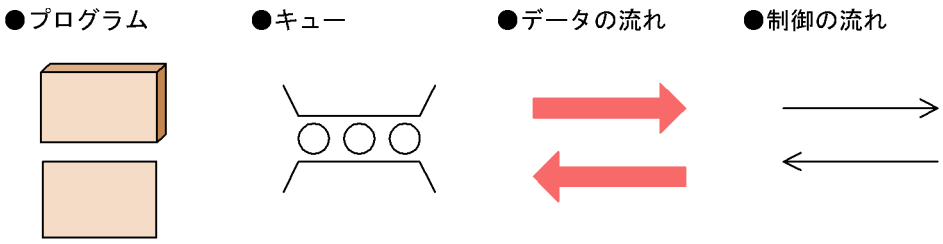
名称	略称
AIX 5L V5.1	AIX
AIX 5L V5.2	
AIX 5L V5.3	

名称	略称
HP-UX 11i V2（IPF）	HP-UX
Java™	Java
JavaBeans™	JavaBeans
Red Hat Enterprise Linux AS 4	Linux
Red Hat Enterprise Linux AS 4（IPF）	
Red Hat Enterprise Linux ES 4（x86）	
Red Hat Enterprise Linux ES 4（AMD64）	
Red Hat Enterprise Linux ES 4 （Intel EM64T）	
Microsoft(R) Visual C++(R)	Visual C++
Microsoft(R) Windows Server(R) 2003, Datacenter Edition	Windows Server 2003
Microsoft(R) Windows Server(R) 2003, Datacenter x64 Edition	
Microsoft(R) Windows Server(R) 2003, Enterprise Edition	
Microsoft(R) Windows Server(R) 2003, Enterprise x64 Edition	
Microsoft(R) Windows Server(R) 2003, Standard Edition	
Microsoft(R) Windows Server(R) 2003, Standard x64 Edition	
Microsoft(R) Windows Vista(R)	Windows Vista
Microsoft(R) Windows(R) XP Professional Operating System	Windows XP
uCosminexus TP1/Message Queue	TP1/Message Queue
uCosminexus TP1/Message Queue（64）	
uCosminexus TP1/Message Queue - Access	TP1/Message Queue - Access
uCosminexus TP1/Message Queue - Access（64）	

- AIX，HP-UX，および Linux を合わせて UNIX と表記することがあります。
- Windows Server 2003，Windows XP，および Windows Vista で機能差がない場合，Windows と表記しています。

■図中で使用する記号

このマニュアルの図中で使用する記号を，次のように定義します。



■文法の記号

このマニュアルで使用する各種の記号を説明します。

属性表示記号

ユーザ指定値の範囲などを説明する記号です。

属性表示記号	意味
～	この記号のあとにユーザ指定値の属性を示します。
《 》	ユーザが指定を省略したときの解釈値を示します。
＜ ＞	ユーザ指定値の構文要素を示します。
(( ))	ユーザ指定値の指定範囲を示します。

構文要素記号

ユーザ指定値の内容を説明する記号です。

構文要素記号	意味
＜符号なし整数＞	数字列（0 ～ 9）
＜ 10 進数＞	数字（0 ～ 9）
＜ 16 進数＞	数字（0 ～ 9）と（A ～ F，a ～ f）
＜識別子＞	先頭がアルファベットの英数字列
＜文字列＞	任意の文字の配列

文法記述記号

記述形式を説明する記号です。

文法記述記号	意味
〔 〕	この記号で囲まれている項目は省略してもよいことを示します。 （例）〔-e 終了処理監視タイマ値〕 -e オプションとそのオペランドを指定するか，何も指定しないことを示します。
{ }	この記号で囲まれている複数の項目のうちから一つを選択することを示します。 （例）{ -p 自システムのポート番号 ｜ -s 自システムのサービス名 } -p 自システムのポート番号と -s 自システムのサービス名のうち，どちらかを指定することを示します。

文法記述記号	意味
| （ストローク）	この記号で区切られた項目は選択できることを示します。 （例）trn_crm_use = Y | N Y または N を指定できることを示します。

■略語一覧

このマニュアルで使用する英略語の一覧を次に示します。

英略語	英字での表記
API	Application Programming Interface
DNS	Domain Name Server
EBCDIC	Extended Binary Coded Decimal Interchange Code
EJB	Enterprise JavaBeans
EUC	Extended UNIX Code
J2EE	Java 2 Platform Enterprise Edition
J2SE	Java 2 Standard Edition
JAR	Java Archive
JIS	Japan Industrial Standard
JMS	Java Message Service
JNDI	Java Naming and Directory Interface
JNI	Java Native Interface
JTA	Java Transaction API
LAN	Local Area Network
MDB	Message-Driven Bean
MQA	Message Queue - Access
MQC	TP1/Message Queue - Access
MQI	Message Queue Interface
OS	Operating System
PTP	Point to Point
RM	Resource Manager
RPC	Remote Procedure Call
SUP	Service Using Program
TCP/IP	Transmission Control Protocol /Internet Protocol
TM	Transaction Manager
TP1/Message Queue	OpenTP1/Message Queue
UAP	User Application Program

英略語	英字での表記
UTF-8	8-bit UCS Transformation Format

■常用漢字以外の漢字の使用について

このマニュアルでは，常用漢字を使用することを基本としていますが，次に示す用語については，常用漢字以外の漢字を使用しています。
宛先（あてさき） 個所（かしょ） 韓国語（かんこくご） 貼り付け（はりつけ） 必須（ひっす）

■ KB（キロバイト）などの単位表記について

1KB（キロバイト），1MB（メガバイト），1GB（ギガバイト），1TB（テラバイト）はそれぞれ 1,024 バイト，$1{,}024^2$ バイト，$1{,}024^3$ バイト，$1{,}024^4$ バイトです。

■謝辞

COBOL 言語仕様は，CODASYL（the Conference on Data Systems Languages：データシステムズ言語協議会）によって，開発された。原開発者に対し謝意を表すとともに，CODASYL の要求に従って以下の謝辞を掲げる。なお，この文章は，COBOL の原仕様書「CODASYL COBOL JOURNAL OF DEVELOPMENT 1984」の謝辞の一部を再掲するものである。

いかなる組織であっても，COBOL の原仕様書とその仕様の全体又は一部分を複製すること，マニュアルその他の資料のための土台として原仕様書のアイデアを利用することは自由である。ただし，その場合には，その刊行物のまえがきの一部として，次の謝辞を掲載しなければならない。書評などに短い文章を引用するときは，"COBOL" という名称を示せば謝辞全体を掲載する必要はない。

COBOL は産業界の言語であり，特定の団体や組織の所有物ではない。
CODASYL　COBOL 委員会又は仕様変更の提案者は，このプログラミングシステムと言語の正確さや機能について，いかなる保証も与えない。さらに，それに関連する責任も負わない。

次に示す著作権表示付資料の著作者及び著作権者

FLOW-MATIC（Sperry Rand Corporation の商標），
Programming for the Univac（R）I and II,
Data Automation Systems，Sperry Rand Corporation　著作権表示 1958 年，1959 年；
IBM Commercial Translator Form No.F 28-8013，IBM 著作権表示 1959 年；
FACT，DSI 27A5260-2760，Minneapolis-Honeywell，著作権表示 1960 年

は，これら全体又は一部分を COBOL の原仕様書中に利用することを許可した。この許可は，COBOL 原仕様書をプログラミングマニュアルや類似の刊行物に複製したり，利用したりする場合にまで拡張される。

目次

1 概要 1

1.1 TP1/Message Queue - Access の概要 2
1.2 TP1/Message Queue - Access の機能 5
1.2.1 MQC クライアント機能 5
1.2.2 MQC サーバ機能 5
1.2.3 トランザクション連携 6
1.2.4 時間監視機能 8
1.2.5 JMS インタフェース機能 10
1.3 Windows を使用する場合の注意事項 11

2 MQC クライアント機能 13

2.1 MQC クライアント機能のセットアップ 14
2.2 MQC クライアント機能の環境設定 22
2.2.1 MQC クライアント機能の環境変数 22
2.2.2 MQC クライアント機能の環境変数のオペランド 28
2.2.3 キュー定義ファイルの作成（JMS インタフェース用） 34
2.2.4 XA インタフェースの設定項目 36
2.2.5 接続先情報定義ファイルの設定項目 38
2.3 クライアントアプリケーションの作成 43
2.3.1 使用できる言語 43
2.3.2 注意事項 44
2.4 MQC クライアント機能の障害対策 47
2.4.1 API トレースファイルの出力形式 48
2.4.2 JavaEnvironment トレースファイルの出力形式 48
2.4.3 JMSAPI トレースファイルの出力形式 52
2.4.4 JMSPRF トレースファイルの取得形式 55

3 MQC クライアント機能の運用コマンド 63

MQC クライアント機能の運用コマンドの概要 64
mqcapiout（API トレースファイル取得） 65

4 MQC クライアント機能の MQI 97

使用できる MQI 98
MQBACK 命令 － ローカルトランザクションのロールバック 100
MQBEGIN 命令 － ローカルトランザクションの開始 103
MQCMIT 命令 － ローカルトランザクションのコミット 106
MQBO 構造体 － ローカルトランザクション開始オプション 109
MQI のサンプルアプリケーション 111
MQI のサンプルコーディング（C 言語） 113
MQI のサンプルコーディング（COBOL 言語） 116

5 MQC クライアント機能の C++ インタフェース 121

MQC クライアント機能の C++ クラス一覧 123
C++ クラス継承図 124
ImqBinary クラス（C++） 125
ImqCache クラス（C++） 128
ImqDeadLetterHeader クラス（C++） 132
ImqDistributionList クラス（C++） 136
ImqError クラス（C++） 138
ImqGetMessageOptions クラス（C++） 140
ImqHeader クラス（C++） 143
ImqItem クラス（C++） 145
ImqMessage クラス（C++） 147
ImqMessageTracker クラス（C++） 155
ImqObject クラス（C++） 159
ImqProcess クラス（C++） 166
ImqPutMessageOptions クラス（C++） 168
ImqQueue クラス（C++） 171
ImqQueueManager クラス（C++） 188
ImqReferenceHeader クラス（C++） 196
ImqString クラス（C++） 200
ImqTrigger クラス（C++） 207
C++ のサンプルアプリケーション 211
C++ のサンプルコーディング 212

6 MQC クライアント機能の Java インタフェース 215
MQC クライアント機能の Java パッケージ 217
MQC クライアント機能の Java クラス一覧 218
Java クラス継承図 219
MQDistributionList クラス（Java） 220
MQDistributionListItem クラス（Java） 222
MQEnvironment クラス（Java） 223
MQException クラス（Java） 225
MQGetMessageOptions クラス（Java） 233
MQManagedObject クラス（Java） 235
MQMessage クラス（Java） 237
MQMessageTracker クラス（Java） 248
MQProcess クラス（Java） 249
MQPutMessageOptions クラス（Java） 251
MQQueue クラス（Java） 253
MQQueueManager クラス（Java） 260
MQC インタフェース（Java） 264
Java のサンプルアプリケーション 276
Java のサンプルコーディング 277

7 MQC クライアント機能の JMS インタフェース 279
JMS インタフェースの Java パッケージ 281
MQC クライアント機能の JMS インタフェース一覧 282
JMS インタフェース継承図 284
JMS インタフェースのメソッドと MQI の対応 286
JMS メッセージのヘッダとプロパティ 287
メッセージセレクタ 292
アプリケーション作成時の注意事項（JMS） 294
アプリケーション使用時の注意事項（JMS） 296
BytesMessage インタフェース（JMS） 298
ConnectionMetaData インタフェース（JMS） 307
DeliveryMode インタフェース（JMS） 310
Destination インタフェース・Queue インタフェース（JMS） 311

Message インタフェース（JMS） 312
MessageConsumer インタフェース・QueueReceiver インタフェース（JMS） 327
MessageProducer インタフェース・QueueSender インタフェース（JMS） 330
QueueBrowser インタフェース（JMS） 338
QueueConnection インタフェース（JMS） 340
QueueConnectionFactory インタフェース（JMS） 344
QueueSession インタフェース（JMS） 346
TemporaryQueue インタフェース（JMS） 354
Enumeration インタフェース（J2SE） 355
MQC インタフェース（JMS） 356
JMS インタフェースのサンプルアプリケーション 361
JMS インタフェースのサンプルコーディング 366

8 メッセージの一覧 377
8.1 メッセージの形式 378
8.1.1 出力形式 378
8.1.2 記述形式 378
8.1.3 メッセージ ID の記号の説明 378
8.2 メッセージ一覧 380

付録 397
付録 A 理由コード 398
付録 B JMS 仕様と MQC クライアント機能の JMS インタフェースとの差異 405
付録 C MQMessage クラスで変換できるコード化文字セット識別子の一覧 412
付録 D 用語解説 414

索引 417

図目次

図 1-1　MQC クライアント機能と MQC サーバ機能の通信　3
図 1-2　XA インタフェース接続　7
図 1-3　JTA インタフェース接続　8
図 1-4　MQC クライアント機能と MQC サーバ機能の監視タイマ　9
図 2-1　接続先情報定義ファイルの設定と解析結果　41
図 4-1　処理の流れ　111
図 5-1　クラス継承図（C++ の場合）　124
図 5-2　処理の流れ（C++ の場合）　211
図 6-1　クラス継承図（Java の場合）　219
図 6-2　処理の流れ（Java の場合）　276
図 7-1　JMS インタフェース継承図　284
図 7-2　例外クラス継承図　285
図 7-3　JMS メッセージと TP1/Message Queue のメッセージとの対応　287
図 7-4　処理の流れ（JMS インタフェースの場合）　362

表目次

表 2-1 MQC クライアント機能のインストール時に作成されるファイル（AIX） 14
表 2-2 MQC クライアント機能のインストール時に作成されるファイル（HP-UX 11i V2.0 (IPF64)） 16
表 2-3 MQC クライアント機能のインストール時に作成されるファイル（Linux） 18
表 2-4 MQC クライアント機能のインストール時に作成されるファイル（Windows） 20
表 2-5 JMS インタフェース使用時に必要な環境変数 25
表 2-6 接続先情報定義ファイルの読み込みの有無による環境変数の設定場所の有効性 26
表 2-7 DCMQCEXPTRN と XA インタフェースの使用の組み合わせによる動作と注意事項 32
表 2-8 RM 関連オブジェクトファイル名 36
表 2-9 リンケージオプション（C 言語） 43
表 2-10 リンケージオプション（COBOL 言語） 43
表 2-11 リンケージオプション（C++ 言語） 44
表 2-12 MQC クライアント機能が障害時に取得する情報 47
表 2-13 JMSAPI トレース情報の出力レベル 53
表 2-14 JMSPRF トレース情報の取得ポイント 56
表 2-15 JMSPRF トレースファイルのイベント ID ごとの取得内容 58
表 3-1 MQC クライアント機能の運用コマンド 64
表 4-1 MQC クライアント機能のクライアントアプリケーションが使用できる MQI 一覧 98
表 4-2 CompCode 引数が MQCC_OK の場合（MQBACK 命令の場合） 100
表 4-3 CompCode 引数が MQCC_WARNING の場合（MQBACK 命令の場合） 101
表 4-4 CompCode 引数が MQCC_FAILED の場合（MQBACK 命令の場合） 101
表 4-5 CompCode 引数が MQCC_OK の場合（MQBEGIN 命令の場合） 104
表 4-6 CompCode 引数が MQCC_FAILED の場合（MQBEGIN 命令の場合） 104
表 4-7 CompCode 引数が MQCC_OK の場合（MQCMIT 命令の場合） 106
表 4-8 CompCode 引数が MQCC_WARNING の場合（MQCMIT 命令の場合） 107
表 4-9 CompCode 引数が MQCC_FAILED の場合（MQCMIT 命令の場合） 107
表 4-10 MQBO 構造体を構成するフィールド一覧 109
表 5-1 クラスの一覧（C++） 123
表 6-1 クラスの一覧（Java） 218
表 6-2 インタフェースの一覧（Java） 218
表 6-3 JavaEnvironment トレース情報の出力レベル 223
表 7-1 JMS インタフェースの一覧 282
表 7-2 インタフェースの一覧（JMS インタフェース） 282

表 7-3　例外クラスの一覧（JMS インタフェース）　283
表 7-4　メソッド発行時に内部で発行される MQI　286
表 7-5　JMS ヘッダと対応する MQMD 構造体のフィールド　288
表 7-6　JMS プロパティと対応する MQMD 構造体のフィールド　288
表 7-7　JMS プロパティの型変換　290
表 8-1　メッセージの種類　379
表 8-2　メッセージの出力先種別　379
表 A-1　理由コード一覧　398
表 B-1　インタフェースの機能差　405
表 B-2　メソッドの機能差　406
表 C-1　MQMessage クラスで変換できるコード化文字セット識別子の一覧　412

1 概要

この章では，TP1/Message Queue - Access（MQC）の接続形態や，MQC クライアント機能と MQC サーバ機能の関係などについて説明します。

1.1 TP1/Message Queue - Access の概要

1.2 TP1/Message Queue - Access の機能

1.3 Windows を使用する場合の注意事項

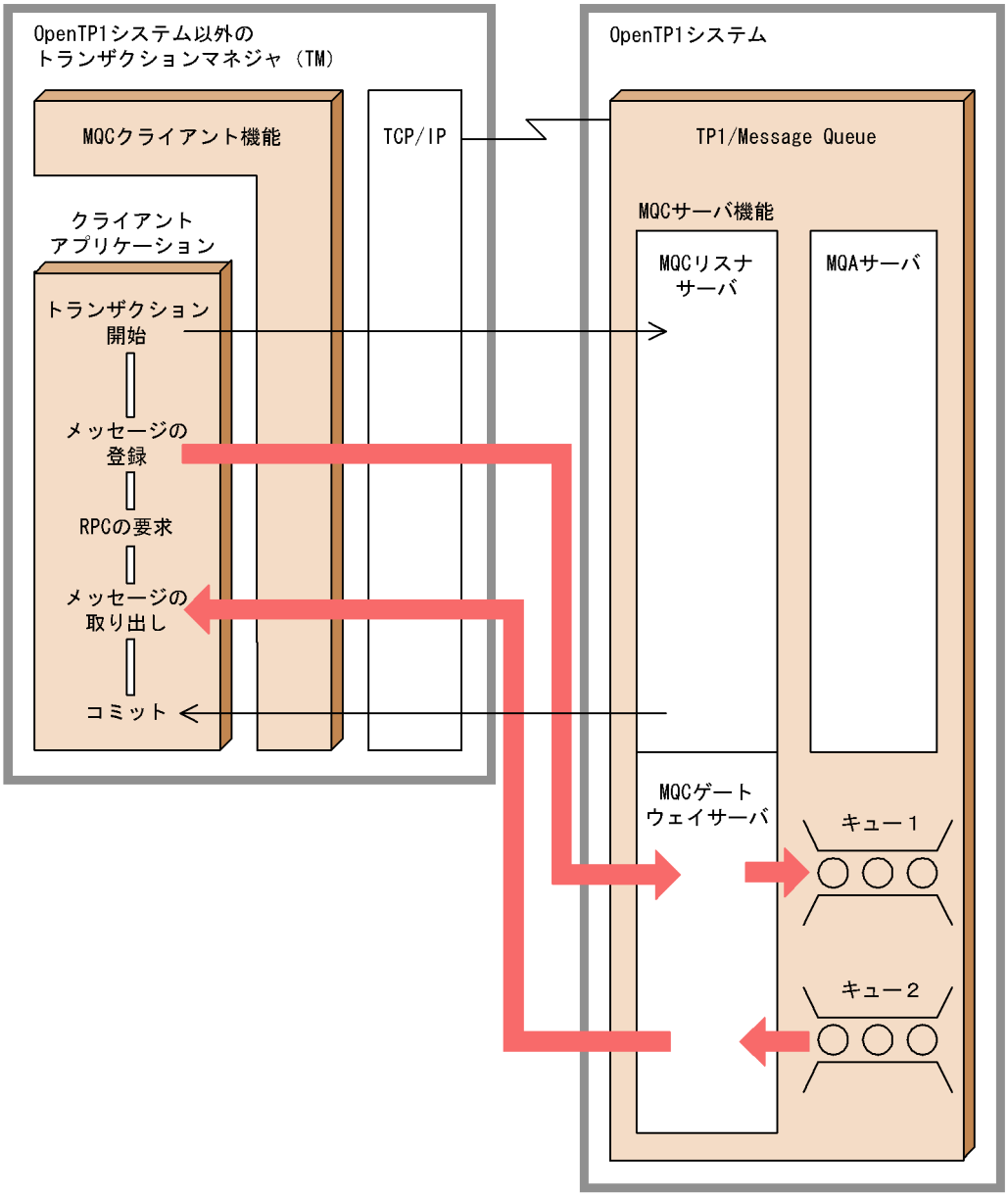
1.1 TP1/Message Queue - Access の概要

TP1/Message Queue - Access（MQC）を使用すると，クライアントアプリケーションから TP1/Message Queue のメッセージキューにメッセージを登録したり，取り出したりできます。

TP1/Message Queue - Access は MQC クライアント機能を提供し，TP1/Message Queue は MQC サーバ機能を提供します。MQC クライアント機能は，OpenTP1 システムが存在しないマシンで動作できます。

MQC クライアント機能と MQC サーバ機能の通信について次の図に示します。

図 1-1　MQC クライアント機能と MQC サーバ機能の通信



TP1/Message Queue - Access の通信では，TCP/IP プロトコル上の MQC **プロトコル**を使用します。

MQC プロトコルは，MQC クライアント機能と MQC サーバ機能との間に MQC **コネクション**を確立します。MQC コネクションの確立は必ず MQC クライアント機能から開始します。MQC コネクションと TCP/IP コネクションとは 1 対 1 の関係です。

MQC プロトコルでは，クライアントアプリケーションで指定されたパラメタを一定の長さに分割してヘッダを付けたあと，TCP/IP に送信します。この分割の単位を**セグメント**といいます。セグメントサイズは，MQC の環境変数 DCMQCSEGSIZE で指定します。

MQC プロトコルにはバージョンがあります。MQC クライアント機能と MQC サーバ機能は同じバージョンで動作します。

1.2 TP1/Message Queue - Access の機能

この節では，TP1/Message Queue - Access の機能について説明します。

1.2.1 MQC クライアント機能

MQC クライアント機能は，ユーザが業務に合わせて作成するクライアントアプリケーションを実行することによって MQC サーバ機能と通信します。

クライアントアプリケーションは，TP1/Message Queue のメッセージキューにメッセージを登録したり，取り出したりします。作成時には，MQI（C 言語または COBOL 言語），C++ 言語，Java 言語，および JMS インタフェースの Java 言語を使用できます。

API はトランザクションの範囲内でも範囲外でも使用できますが，トランザクションと同期を取る場合は，接続する TM が X/Open の XA インタフェースに準拠している必要があります。

1.2.2 MQC サーバ機能

MQC サーバ機能は，MQC リスナサーバと MQC ゲートウェイサーバとで構成されます。なお，MQC サーバ機能を構成するファイルは，TP1/Message Queue に添付されています。MQC サーバ機能の定義，運用，および障害対策については，マニュアル「TP1/Message Queue 使用の手引」を参照してください。

● **MQC リスナサーバ**

MQC リスナサーバは，TP1/Message Queue に組み込まれて動作します。MQC クライアント機能からの通信を基に MQC ゲートウェイサーバを予約し，トランザクションを処理します。

● **MQC ゲートウェイサーバ**

MQC ゲートウェイサーバは，OpenTP1 システムのユーザサーバとして動作します。MQC ゲートウェイサーバは，MQC クライアント機能とは 1 対 1 の関係でキューアクセスを処理します。また，送受信するメッセージを格納するために TP1/Message Queue のキューを使用します。

注意事項

TP1/Message Queue 07-00 より前のバージョンの MQC サーバに接続しようとした場合，MQC ゲートウェイサーバが異常終了（アボートコード：Mqcgw0c），または MQCONN 命令が MQRC_Q_MGR_NOT_AVAILABLE（2059）でエラーリターンします。必ず，TP1/Message Queue 07-00 以降の MQC サーバに接続してください。

1.2.3 トランザクション連携

TP1/Message Queue - Access のトランザクション連携には，次の 3 種類があります。

● XA インタフェース接続

● 非 XA インタフェース接続

● JTA インタフェース接続

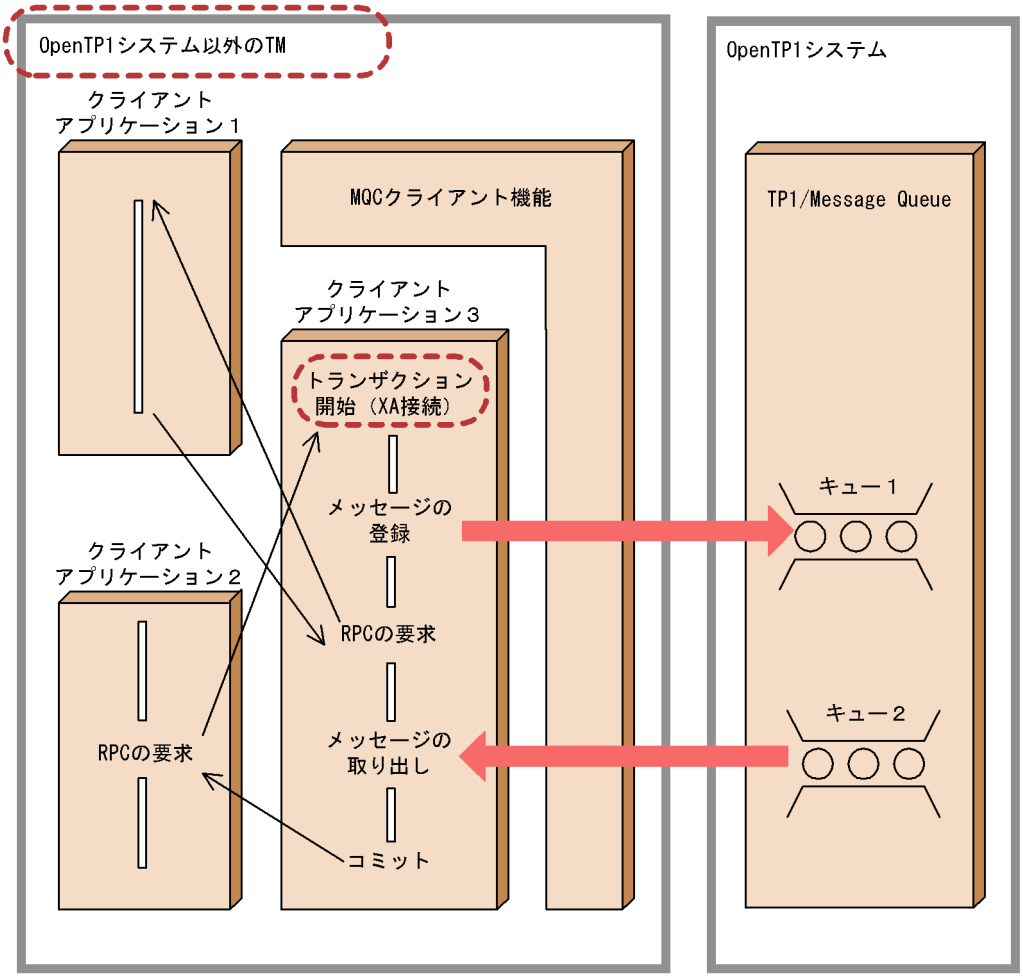
（1）XA インタフェース接続と非 XA インタフェース接続

XA インタフェース接続では，MQC クライアント機能が OpenTP1 システム以外の TM（例：TPBroker）と接続して動作します。

一方，非 XA インタフェース接続は，MQC クライアント機能が OpenTP1 システム以外の TM と接続しないで動作する形態です。

XA インタフェース接続について次の図に示します。

図 1-2 XA インタフェース接続



非 XA インタフェース接続の場合は，図 1-2 の点線で囲んだ部分が XA インタフェース接続の場合と異なります。つまり，次に示す 2 点が各接続形態の相違点です。

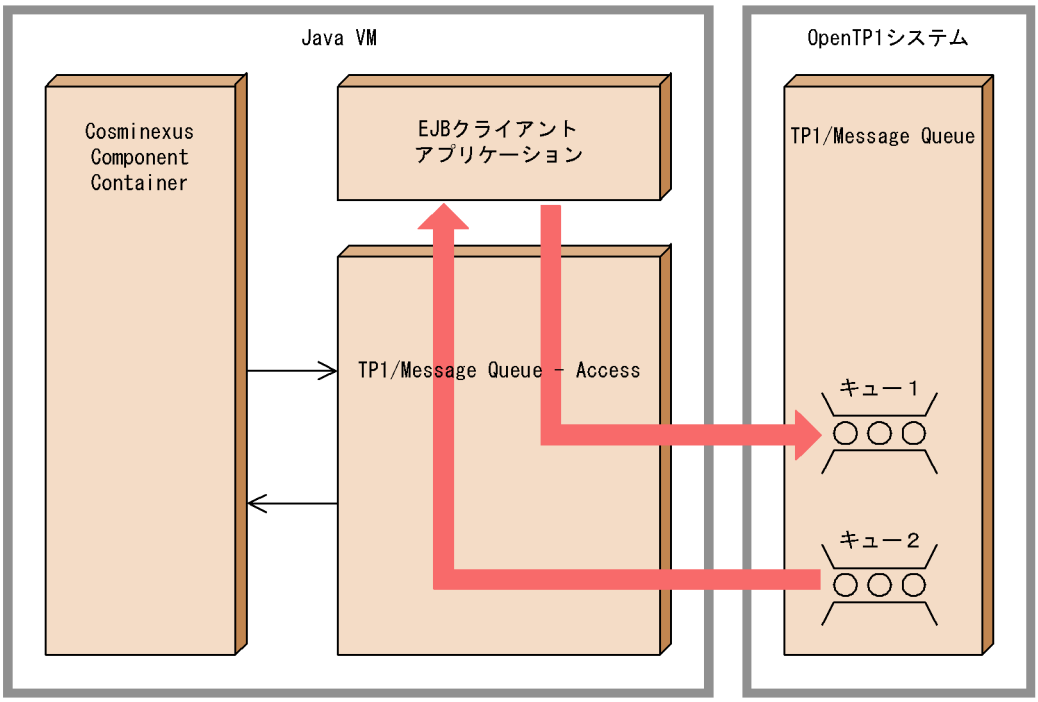
- 非 XA インタフェース接続では，クライアントアプリケーションおよび MQC クライアント機能が OpenTP1 以外の TM に含まれません。
- 非 XA インタフェース接続のクライアントアプリケーション 3 でトランザクションを開始するときは，「XA 接続」で開始するのではなく「ローカルトランザクション」を開始します。ローカルトランザクションは，参加プログラムがキューマネジャだけのトランザクションです。ローカルトランザクションではキューマネジャが TM として機能します。

(2) JTA インタフェース接続

JTA インタフェース接続では，Cosminexus Component Container を TM として接続し

ます。この接続は，Cosminexus 側の EJB クライアントアプリケーションから発行される JMS インタフェースの API を MQC クライアント機能が持つ JNI の MQI に変換することで実現します。変換された MQI は，MQC プロトコルを介して TP1/Message Queue に通知されます。JTA インタフェース接続について次の図に示します。

図 1-3　JTA インタフェース接続

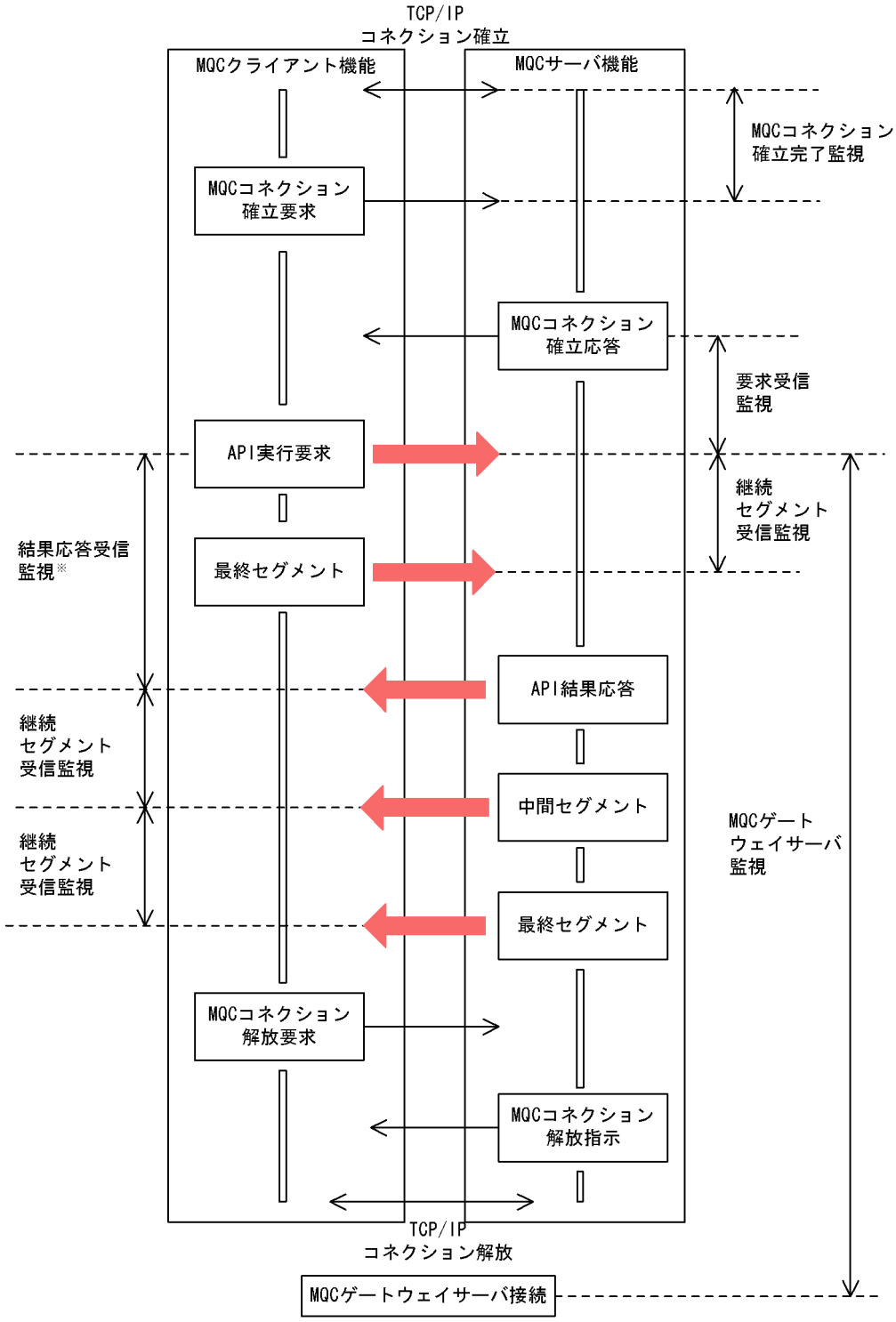


1.2.4　時間監視機能

MQC クライアント機能と MQC サーバ機能の**監視タイマ**について次の図に示します。

MQC クライアント機能の監視タイマ値の設定については，「2.2　MQC クライアント機能の環境設定」を参照してください。MQC サーバ機能の監視タイマ値の設定については，マニュアル「TP1/Message Queue 使用の手引」を参照してください。

図 1-4　MQC クライアント機能と MQC サーバ機能の監視タイマ



注※

この時間は，DCMQCTIMEREQ オペランドで指定する結果応答受信監視タイマ値となります。結果応答受信監視タイマ値には，DCMQCTIMEGET オペランドで指定する，MQGET 命令の待ち合わせ最大時間を含んでいます。

1.2.5 JMS インタフェース機能

JMS インタフェース機能は，EJB クライアントアプリケーションおよび Cosminexus Component Container を TP1/Message Queue に接続します。この場合，JTA によってトランザクションと連携します。

（1） コネクション制御

JMS インタフェース機能では，Cosminexus による**コネクションプーリング**でコネクションを制御します。コネクションプーリングでは，クライアントアプリケーションがコネクションのクローズを要求すると，内部でコネクションの確立状態が保持されたままとなります。その後，新しいコネクションの確立を要求すると，確立状態が保持されていたコネクションが割り当てられます。これによって，コネクションの確立および解放に関するオーバーヘッドが減少します。

（2） Message-Driven Bean キュー監視機能

Message-Driven Bean は，Cosminexus Component Container の EJB コンテナが実行を制御する Enterprise Bean の一つで，JMS と連携するメッセージ駆動タイプの Bean です。TP1/Message Queue - Access では，Message-Driven Bean を介して，キューに到着したメッセージを EJB クライアントアプリケーションに割り当てます。

Message-Driven Bean キュー監視機能の動作の仕組みは次のとおりです。

1. TP1/Message Queue - Access は，複数のキュー監視スレッドを起動して常に TP1/Message Queue のキューへのメッセージ到着を監視します。
2. キューにメッセージが到着すると，キュー監視スレッドはそのメッセージを取得して Message-Driven Bean に通知します。
3. Message-Driven Bean は，メッセージを処理する EJB クライアントアプリケーションを決定して起動します。
4. EJB クライアントアプリケーションは，データベースなどのほかのリソースマネジャ（RM）にアクセスしてメッセージを処理します。

1.3 Windows を使用する場合の注意事項

Windows の TP1/Message Queue - Access を使用する場合，次に示す注意事項があります。

- Windows の環境変数の表記は，$DCDIR ではなく %DCDIR% になります。
- 複数のパス名の区切り文字が，UNIX と Windows とで異なります。UNIX ではコロン（：）が複数のパス名の区切り文字ですが，Windows ではコロンはドライブ名とディレクトリ名との区切り文字になります。Windows ではパス名の区切り文字にセミコロン（；）を使用してください。
- パス名を完全パスで指定する際には，必ずドライブ名を記述してください。
- OpenTP1 は大文字と小文字を区別します。コマンドのオプションや，定義ファイルに記述した文字列をコマンド引数で使用するような場合は，注意してください。
  例えば，-a オプションと -A オプションでは異なるオプションを表します。
- ファイル名を指定するときは，ドライブ名以外にコロン（：）を含んだファイル名を指定しないでください。

2 MQC クライアント機能

この章では，MQC クライアント機能の定義，クライアントアプリケーションの作成，および障害対策について説明します。

2.1 MQC クライアント機能のセットアップ

2.2 MQC クライアント機能の環境設定

2.3 クライアントアプリケーションの作成

2.4 MQC クライアント機能の障害対策

2.1 MQC クライアント機能のセットアップ

MQC クライアント機能のライブラリのディレクトリを，適用 OS に応じて次に示す環境変数に追加してください。

- AIX の場合：LIBPATH
- HP-UX の場合：SHLIB_PATH
- Linux の場合：LD_LIBRARY_PATH
- Windows の場合：LIB
  Windows の場合，環境変数 PATH に " インストールフォルダ ¥bin" も追加してください。

表 2-1 ～表 2-4 に，MQC クライアント機能のインストール時に作成されるファイルを適用 OS ごとに示します。なお，表中の `$MQCDIR`，および `%MQCDIR%` は TP1/Message Queue - Access のファイルが格納されているディレクトリです。標準はインストールディレクトリとなります。

表 2-1 MQC クライアント機能のインストール時に作成されるファイル（AIX）

言語	名称	ディレクトリ	ファイル名
C	XA インタフェースライブラリ※	`$MQCDIR/lib`	`libmqcx.a`
	非 XA インタフェースライブラリ※	`$MQCDIR/lib`	`libmqc.a`
	ヘッダファイル	`$MQCDIR/include`	`cmqc.h` `cmqcfc.h`
	サンプル AP	`$MQCDIR/examples/c`	`mqcsample.c` `makefile`
COBOL	XA インタフェースライブラリ※	`$MQCDIR/lib`	`libmqccbx.a`
	非 XA インタフェースライブラリ※	`$MQCDIR/lib`	`libmqccb.a`
	COBOL コピーファイル	`$MQCDIR/include/COBOL`	`CMQ*.cbl`

	サンプル AP	`$MQCDIR/examples/cobol`	`MQCSAMPLE.cbl`
			`makefile`
			`makefile2002`

C++	非 XA インタフェースライブラリ※	`$MQCDIR/lib`	`libmqccpp.a`
	ヘッダファイル	`$MQCDIR/include/cplus`	`imq*.hpp` `imqtype.h`
	サンプル AP	`$MQCDIR/examples/cpp`	`mqcsample.cpp` `makefile`

言語	名称	ディレクトリ	ファイル名
Java	非 XA インタフェースライブラリ※	$MQCDIR/lib	libmqcj.a
	Java クラスライブラリ	$MQCDIR/lib	mqc.jar
	サンプル AP	$MQCDIR/examples/java	MQAccessSample.java makefile
Java の JMS インタフェース	Java クラスライブラリ	$MQCDIR/lib	mqcadpt.rar mqcadptdef.jar
	サンプル AP	$MQCDIR/examples/jms/SessionBean1	JMSSample1.java JMSSample1.Client.java JMSSample1EJB.java JMSSample1Home.java compileBean compileClient testClient QueueConfig config.xml deployApp unDeployApp
		$MQCDIR/examples/jms/SessionBean1/DD/META-INF	ejb-jar.xml application.xml
	サンプル AP（MQC インタフェース使用）	$MQCDIR/examples/jms/SessionBean2	JMSSample2.java JMSSample2.Client.java JMSSample2EJB.java JMSSample2Home.java compileBean compileClient testClient QueueConfig config.xml deployApp unDeployApp
		$MQCDIR/examples/jms/SessionBean2/DD/META-INF	ejb-jar.xml application.xml
その他	運用コマンド	$MQCDIR/bin	mqcapiout

注※

これらのライブラリは，共用ライブラリとしても使用できます。

表 2-2 MQC クライアント機能のインストール時に作成されるファイル（HP-UX 11i V2.0（IPF64））

言語	名称	ディレクトリ	ファイル名
C	非 XA インタフェースライブラリ	$MQCDIR/lib	libmqc.a
	非 XA インタフェース共用ライブラリ	$MQCDIR/lib	libmqc.so
	ヘッダファイル	$MQCDIR/include	cmqc.h cmqcfc.h
	サンプル AP	$MQCDIR/examples/c	mqcsample.c makefile
C++	非 XA インタフェースライブラリ	$MQCDIR/lib	libmqccpp.a
	非 XA インタフェース共用ライブラリ	$MQCDIR/lib	libmqccpp.so
	ヘッダファイル	$MQCDIR/include/cplus	imq*.hpp imqtype.h
	サンプル AP	$MQCDIR/examples/cpp	mqcsample.cpp makefile
Java	非 XA インタフェース共用ライブラリ	$MQCDIR/lib	libmqcj.so
	Java クラスライブラリ	$MQCDIR/lib	mqc.jar
	サンプル AP	$MQCDIR/examples/java	MQAccessSample.java makefile

言語	名称	ディレクトリ	ファイル名
Java の JMS インタフェース	Java クラスライブラリ	$MQCDIR/lib	mqcadpt.rar mqcadptdef.jar
	サンプル AP	$MQCDIR/examples/jms/SessionBean1	JMSSample1.java JMSSample1.Client.java JMSSample1EJB.java JMSSample1Home.java compileBean compileClient testClient QueueConfig config.xml deployApp unDeployApp
		$MQCDIR/examples/jms/SessionBean1/DD/META-INF	ejb-jar.xml application.xml
	サンプル AP（MQC インタフェース使用）	$MQCDIR/examples/jms/SessionBean2	JMSSample2.java JMSSample2.Client.java JMSSample2EJB.java JMSSample2Home.java compileBean compileClient testClient QueueConfig config.xml deployApp unDeployApp
		$MQCDIR/examples/jms/SessionBean2/DD/META-INF	ejb-jar.xml application.xml
その他	運用コマンド	$MQCDIR/bin	mqcapiout

表 2-3 MQC クライアント機能のインストール時に作成されるファイル（Linux）

言語	名称	ディレクトリ	ファイル名
C	XA インタフェースライブラリ※1	$MQCDIR/lib	libmqcx.a
	XA インタフェース共用ライブラリ※1	$MQCDIR/lib	libmqcx.so
	非 XA インタフェースライブラリ	$MQCDIR/lib	libmqc.a
	非 XA インタフェース共用ライブラリ	$MQCDIR/lib	libmqc.so
	ヘッダファイル	$MQCDIR/include	cmqc.h cmqcfc.h
	サンプル AP	$MQCDIR/examples/c	mqcsample.c makefile
COBOL※2	XA インタフェースライブラリ	$MQCDIR/lib	libmqccbx.a
	XA インタフェース共用ライブラリ	$MQCDIR/lib	libmqccbx.so
	非 XA インタフェースライブラリ	$MQCDIR/lib	libmqccb.a
	非 XA インタフェース共用ライブラリ	$MQCDIR/lib	libmqccb.so
	COBOL コピーファイル	$MQCDIR/include/COBOL	CMQ*.cbl
	サンプル AP	$MQCDIR/examples/cobol	MQCSAMPLE.cbl makefile
C++	非 XA インタフェースライブラリ	$MQCDIR/lib	libmqccpp.a
	非 XA インタフェース共用ライブラリ	$MQCDIR/lib	libmqccpp.so
	ヘッダファイル	$MQCDIR/include/cplus	imq*.hpp imqtype.h
	サンプル AP	$MQCDIR/examples/cpp	mqcsample.cpp makefile
Java	非 XA インタフェース共用ライブラリ	$MQCDIR/lib	libmqcj.so
	Java クラスライブラリ	$MQCDIR/lib	mqc.jar
	サンプル AP	$MQCDIR/examples/java	MQAccessSample.java makefile

言語	名称	ディレクトリ	ファイル名
Java の JMS インタフェース	Java クラスライブラリ	$MQCDIR/lib	mqcadpt.rar mqcadptdef.jar
	サンプル AP	$MQCDIR/examples/jms/SessionBean1	JMSSample1.java JMSSample1.Client.java JMSSample1EJB.java JMSSample1Home.java compileBean compileClient config.xml deployApp testClient QueueConfig unDeployApp
		$MQCDIR/examples/jms/SessionBean1/DD/META-INF	ejb-jar.xml application.xml
	サンプル AP（MQC インタフェース使用）	$MQCDIR/examples/jms/SessionBean2	JMSSample2.java JMSSample2.Client.java JMSSample2EJB.java JMSSample2Home.java compileBean compileClient config.xml deployApp testClient QueueConfig unDeployApp
		$MQCDIR/examples/jms/SessionBean2/DD/META-INF	ejb-jar.xml application.xml
その他	運用コマンド	$MQCDIR/bin	mqcapiout

注※ 1

IPF64 では，C の XA インタフェースが使用できません。

注※ 2

Red Hat Enterprise Linux 4 上にはインストールされません。

表 2-4 MQC クライアント機能のインストール時に作成されるファイル（Windows）

言語	名称	フォルダ	ファイル名
C	XA インタフェースライブラリ	%MQCDIR%¥bin	libmqcx.dll
		%MQCDIR%¥lib	libmqcx.lib
	非 XA インタフェースライブラリ	%MQCDIR%¥bin	libmqc.dll
		%MQCDIR%¥lib	libmqc.lib
	サンプル AP	%MQCDIR%¥examples¥c	mqcsample.c makefile
	ヘッダファイル	%MQCDIR%¥include	cmqc.h cmqcfc.h
COBOL	XA インタフェースライブラリ	%MQCDIR%¥bin	libmqccbx.dll
		%MQCDIR%¥lib	libmqccbx.lib
	非 XA インタフェースライブラリ	%MQCDIR%¥bin	libmqccb.dll
		%MQCDIR%¥lib	libmqccb.lib
	COBOL コピーファイル	%MQCDIR%¥include¥COBOL	CMQ*.cbl
	サンプル AP	%MQCDIR%¥examples¥cobol	MQCSAMPLE.cbl makefile
C++	XA インタフェースライブラリ	%MQCDIR%¥bin	libmqcppx.dll
		%MQCDIR%¥lib	libmqcppx.lib
	非 XA インタフェースライブラリ	%MQCDIR%¥bin	libmqcpp.dll
		%MQCDIR%¥lib	libmqcpp.lib
	ヘッダファイル	%MQCDIR%¥include¥cplus	imq*.hpp imqtype.h
	サンプル AP	%MQCDIR%¥examples¥cpp	mqcsample.cpp makefile
Java	非 XA インタフェース共用ライブラリ	%MQCDIR%¥bin	mqcj.dll
	非 XA インタフェース Java クラスライブラリ	%MQCDIR%¥lib	mqc.jar
	サンプル AP	%MQCDIR%¥examples¥java	MQAccessSample.java makefile

言語	名称	フォルダ	ファイル名
Java の JMS インタフェース	Java クラスライブラリ	%MQCDIR%¥lib	mqcadpt.rar mqcadptdef.jar
	サンプル AP	%MQCDIR%¥examples¥jms¥SessionBean1	JMSSample1.java JMSSample1Client.java JMSSample1EJB.java JMSSample1Home.java compileBean.bat compileClient.bat testClient.bat QueueConfig deployApp unDeployApp config.xml
		%MQCDIR%¥examples¥jms¥SessionBean1¥DD¥META-INF	ejb-jar.xml application.xml
	サンプル AP （MQC インタフェース使用）	%MQCDIR%¥examples¥jms¥SessionBean2	JMSSample2.java JMSSample2Client.java JMSSample2EJB.java JMSSample2Home.java compileBean.bat compileClient.bat testClient.bat QueueConfig deployApp unDeployApp config.xml
		%MQCDIR%¥examples¥jms¥SessionBean2¥DD¥META-INF	ejb-jar.xml application.xml
その他	運用コマンド	%MQCDIR%¥bin	mqcapiout.exe
	readme ファイル	%MQCDIR%¥readme	readme.txt

2.2 MQC クライアント機能の環境設定

MQC クライアント機能が使用する環境変数，Java 言語の JMS インタフェースを使用する場合に必要なキュー定義ファイル，および XA インタフェースの設定について説明します。

2.2.1 MQC クライアント機能の環境変数

UNIX の場合，MQC クライアント機能が使用する環境変数の設定形式は，MQC クライアント機能があるコンピュータのシェルによって異なります。

- ボーンシェルを使用する場合は /etc/profile または $HOME/.profile に記述してください。
- C シェルを使用する場合は /etc/cshrc または $HOME/.cshrc に記述してください。

Windows の場合，コントロールパネルを起動して環境変数を設定してください。

Java 言語の JMS インタフェースを使用する場合は，uCosminexus Application Server 上でリソースアダプタのプロパティ定義に次の環境変数を設定してください。

(1) ボーンシェルを使用する場合の形式（UNIX）

```
$ DCMQCSRVHOSTNAME = 窓口となるMQCリスナサーバのホスト名
$ DCMQCSRVHOSTIP = 窓口となるMQCリスナサーバのIPアドレス
$ DCMQCSRVSERVNAME = 窓口となるMQCリスナサーバのサービス名
$ DCMQCSRVPORT = 窓口となるMQCリスナサーバのポート番号
〔$ DCMQCCLTHOSTIP = 自システムのIPアドレス〕
〔$ DCMQCCLTPORT = 自システムのポート番号〕
〔$ DCMQCTIMEREQ = MQCリスナサーバまたはMQCゲートウェイサーバからの
                   結果応答受信監視タイマ値〕
〔$ DCMQCTIMEGET = MQCゲートウェイサーバでのMQGET命令の
                   待ち合わせ最大時間〕
〔$ DCMQCTIMESEG = MQCリスナサーバまたはMQCゲートウェイサーバからの
                   継続セグメント受信監視タイマ値〕
〔$ DCMQCTIMECON = MQCリスナサーバまたはMQCゲートウェイサーバへの
                   最大接続試行時間〕
〔$ DCMQCSEGSIZE = 転送時のセグメントサイズ〕
〔$ DCMQCAPITRCOUT = APIトレースディスク出力要否〕
〔$ DCMQCAPITRCFILNUM = APIトレースファイル数〕
〔$ DCMQCAPITRCSIZE = APIトレース容量（エントリ数）〕
〔$ DCMQCAPITRCWRITE = APIトレース出力単位（エントリ数）〕
〔$ DCMQCEXPTRN = トランザクション処理方式の拡張要否〕
〔$ DCMQCDEFCON = 接続先情報定義ファイル読み込み要否〕
〔$ DCMQCDEFCONPATH = 接続先情報定義ファイルのパス〕

$ export DCMQCSRVHOSTNAME DCMQCSRVHOSTIP DCMQCSRVSERVNAME
DCMQCSRVPORT DCMQCCLTHOSTIP DCMQCCLTPORT DCMQCTIMEREQ
DCMQCTIMEGET DCMQCTIMESEG DCMQCTIMECON DCMQCSEGSIZE
DCMQCAPITRCOUT DCMQCAPITRCFILNUM DCMQCAPITRCSIZE DCMQCAPITRCWRITE
```

```
DCMQCEXPTRN DCMQCDEFCON DCMQCDEFCONPATH
```

（2） C シェルを使用する場合の形式（UNIX）

```
% setenv DCMQCSRVHOSTNAME　窓口となるMQCリスナサーバのホスト名
% setenv DCMQCSRVHOSTIP　窓口となるMQCリスナサーバのIPアドレス
% setenv DCMQCSRVSERVNAME　窓口となるMQCリスナサーバのサービス名
% setenv DCMQCSRVPORT　窓口となるMQCリスナサーバのポート番号
〔% setenv DCMQCCLTHOSTIP　自システムのIPアドレス〕
〔% setenv DCMQCCLTPORT　自システムのポート番号〕
〔% setenv DCMQCTIMEREQ　MQCリスナサーバまたはMQCゲートウェイサーバ
                          からの結果応答受信監視タイマ値〕
〔% setenv DCMQCTIMEGET　MQCゲートウェイサーバでのMQGET命令の
                          待ち合わせ最大時間〕
〔% setenv DCMQCTIMESEG　MQCリスナサーバまたはMQCゲートウェイサーバ
                          からの継続セグメント受信監視タイマ値〕
〔% setenv DCMQCTIMECON = MQCリスナサーバまたはMQCゲートウェイサーバへの
                           最大接続試行時間〕
〔% setenv DCMQCSEGSIZE　転送時のセグメントサイズ〕
〔% setenv DCMQCAPITRCOUT　APIトレースディスク出力要否〕
〔% setenv DCMQCAPITRCFILNUM　APIトレースファイル数〕
〔% setenv DCMQCAPITRCSIZE　APIトレース容量（エントリ数）〕
〔% setenv DCMQCAPITRCWRITE　APIトレース出力単位（エントリ数）〕
〔% setenv DCMQCEXPTRN　トランザクション処理方式の拡張要否〕
〔% setenv DCMQCDEFCON　接続先情報定義ファイル読み込み要否〕
〔% setenv DCMQCDEFCONPATH　接続先情報定義ファイルのパス〕
```

（3） Windows での形式

```
DCMQCSRVHOSTNAME　窓口となるMQCリスナサーバのホスト名
DCMQCSRVHOSTIP　窓口となるMQCリスナサーバのIPアドレス
DCMQCSRVSERVNAME　窓口となるMQCリスナサーバのサービス名
DCMQCSRVPORT　窓口となるMQCリスナサーバのポート番号
〔DCMQCCLTHOSTIP　自システムのIPアドレス〕
〔DCMQCCLTPORT　自システムのポート番号〕
〔DCMQCTIMEREQ　MQCリスナサーバまたはMQCゲートウェイサーバからの
                 結果応答受信監視タイマ値〕
〔DCMQCTIMEGET　MQCゲートウェイサーバでのMQGET命令の
                 待ち合わせ最大時間〕
〔DCMQCTIMESEG　MQCリスナサーバまたはMQCゲートウェイサーバからの
                 継続セグメント受信監視タイマ値〕
〔DCMQCTIMECON　MQCリスナサーバまたはMQCゲートウェイサーバへの
                 最大接続試行時間〕
〔DCMQCSEGSIZE　転送時のセグメントサイズ〕
〔DCMQCAPITRCOUT　APIトレースディスク出力要否〕
〔DCMQCAPITRCFILNUM　APIトレースファイル数〕
〔DCMQCAPITRCSIZE　APIトレース容量（エントリ数）〕
〔DCMQCAPITRCWRITE　APIトレース出力単位（エントリ数）〕
〔DCMQCEXPTRN　トランザクション処理方式の拡張要否〕
〔DCMQCDEFCON = 接続先情報定義ファイル読み込み要否〕
〔DCMQCDEFCONPATH = 接続先情報定義ファイルのパス〕
```

（4） JMS インタフェース使用時の環境変数の設定

JMS インタフェースを使用する場合は，(1) ～ (3) で示した MQC クライアント機能の環境変数に加えて，JMS インタフェース専用の環境変数を設定する必要があります。この

設定では，uCosminexus Application Server でリソースアダプタのプロパティ定義に設定します。詳細については，マニュアル「Cosminexus アプリケーション設定操作ガイド」を参照してください。

(a) デプロイおよびアンデプロイ時の注意事項（JMS インタフェース使用時）

- TP1/Message Queue - Access をインポートするには，リソースアダプタのインポートで mqcadpt.rar ファイルを指定します。mqcadpt.rar ファイルは，TP1/Message Queue - Access のインストール先の lib ディレクトリに格納されています。
- TP1/Message Queue - Access は，アプリケーションデプロイして使用することはできません。また，複数デプロイして使用することもできません。
- TP1/Message Queue - Access が出力するメッセージは，すべてログファイルに書き込まれます。ログを出力するかどうかは，「リソースアダプタのプロパティ定義」－「実行時のプロパティ」で設定できます。設定する場合は，必ずログを出力する設定にしてください。
- TP1/Message Queue - Access をデプロイしたあとに削除する場合は，サーバを再起動してから削除してください。
- TP1/MessageQueue - Access のリプレースなどで，TP1/Message Queue - Access を削除する場合は，削除する前に必ず TP1/Message Queue - Access をエクスポートしてください。エクスポートの操作については，マニュアル「Cosminexus アプリケーション設定操作ガイド」を参照してください。
  なお，TP1/Message Queue - Access をスタートしたあとに TP1/Message Queue - Access を削除するとエラーになります。この場合は，サーバを再起動してから TP1/Message Queue - Access を削除してください。

(b) 環境変数の設定項目（JMS インタフェース使用時）

JMS インタフェースを使用する場合に設定する必要がある環境変数を次の表に示します。

表 2-5 JMS インタフェース使用時に必要な環境変数

分類	設定項目	設定する内容
MQC クライアント機能共通	DCMQCSRVHOSTNAME	窓口となる MQC リスナサーバのホスト名
	DCMQCSRVHOSTIP	窓口となる MQC リスナサーバの IP アドレス
	DCMQCSRVSERVNAME	窓口となる MQC リスナサーバのサービス名
	DCMQCSRVPORT	窓口となる MQC リスナサーバのポート番号
	DCMQCCLTHOSTIP	自システムの IP アドレス
	DCMQCCLTPORT	自システムのポート番号
	DCMQCTIMEREQ	MQC リスナサーバまたは MQC ゲートウェイサーバからの結果応答受信監視タイマ値
	DCMQCTIMEGET	MQC ゲートウェイサーバでの MQGET 命令の待ち合わせ最大時間
	DCMQCTIMESEG	MQC リスナサーバまたは MQC ゲートウェイサーバからの継続セグメント受信監視タイマ値
	DCMQCTIMECON	MQC リスナサーバまたは MQC ゲートウェイサーバへの最大接続試行時間
	DCMQCSEGSIZE	転送時のセグメントサイズ
	DCMQCAPITRCOUT	API トレースディスク出力要否
	DCMQCAPITRCFILNUM	API トレースファイル数
	DCMQCAPITRCSIZE	API トレース容量（エントリ数）
	DCMQCAPITRCWRITE	API トレース出力単位（エントリ数）
JMS インタフェース専用	ModelQueueName	TemporaryQueue オブジェクト生成時のモデルキュー定義
	TraceLevel	JMSAPI トレース情報の出力レベル
	PrfTraceLevel	JMSPRF トレース情報の取得レベル
	QueueConfigFileName	キュー定義ファイル名

注

JMS インタフェースを使用する場合の MQC クライアント機能の環境変数の型は次のとおりです。

- TraceLevel および PrfTraceLevel：java.lang.Integer 型
- その他の環境変数：すべて java.lang.String 型

（5） TP1/Server Base と連携する場合の注意事項

● TP1/Server Base と連携する場合，システム環境定義の $DCCONFPATH/env に次の形式で設定してください。

```
putenv DCMQCSRVHOSTIP nnn.nnn.nnn.nnn
putenv DCMQCSRVPORT nnnnn
```

詳細については，「2.2.2(1) 環境変数のオペランド（共通）」の各オペランドの説明を参照してください。

- XA 連携を行う場合，ユーザサービス定義で次の設定をしてください。
  dc_rpc_open 関数発行時に xa_open 関数を発行し，dc_rpc_close 関数発行時に xa_close 関数を発行するための設定となります。

```
set trn_rm_open_close_scope=process
```

詳細については，マニュアル「OpenTP1 システム定義」を参照してください。また，TP1/Message Queue - Access 07-00 以降をインストールする前に旧バージョンをアンインストールしてください。

(6) 環境変数の設定場所の有効性

接続先情報定義ファイルを読み込むかどうかによって，有効になる環境変数の設定場所が異なります。接続先情報定義ファイルを読み込む（DCMQCDEFCON=Y）場合，および接続先情報定義ファイルを読み込まない（DCMQCDEFCON=N）場合の，環境変数の設定場所の有効性を次の表に示します。

表 2-6 接続先情報定義ファイルの読み込みの有無による環境変数の設定場所の有効性

環境変数名	指定する内容	DCMQCDEFCON=Y		DCMQCDEFCON=N	
		接続先情報定義ファイル	OS に設定する環境変数	接続先情報定義ファイル	OS に設定する環境変数
DCMQCDEFCON	接続先情報定義ファイル読み込み要否	−	○	−	○
DCMQCDEFCONPATH	接続先情報定義ファイルのパス	−	○	−	−
DCMQCMGRNAME	接続先の MQ サーバのキューマネジャ名	○	−	−	−
DCMQCSRVHOSTNAME	窓口となる MQC リスナサーバのホスト名	○	−	−	○
DCMQCSRVHOSTIP	窓口となる MQC リスナサーバの IP アドレス	○	−	−	○
DCMQCSRVSERVNAME	窓口となる MQC リスナサーバのサービス名	○	−	−	○
DCMQCSRVPORT	窓口となる MQC リスナサーバのポート番号	○	−	−	○

環境変数名	指定する内容	DCMQCDEFCON=Y		DCMQCDEFCON=N	
		接続先情報定義ファイル	OS に設定する環境変数	接続先情報定義ファイル	OS に設定する環境変数
DCMQCCLTHOSTIP	自システムの IP アドレス	○	－	－	○
DCMQCCLTPORT	自システムのポート番号	○	－	－	○
DCMQCTIMEREQ	MQC リスナサーバまたは MQC ゲートウェイサーバからの結果応答受信監視タイマ値	○	－	－	○
DCMQCTIMEGET	MQC ゲートウェイサーバでの MQGET 命令の待ち合わせ最大時間	○	－	－	○
DCMQCTIMESEG	MQC リスナサーバまたは MQC ゲートウェイサーバからの継続セグメント受信監視タイマ値	○	－	－	○
DCMQCTIMECON	MQC リスナサーバまたは MQC ゲートウェイサーバへの最大接続試行時間	○	－	－	○
DCMQCSEGSIZE	転送時のセグメントサイズ	○	－	－	○
DCMQCAPITRCOUT	API トレースディスク出力要否	－	○	－	○
DCMQCAPITRCFILNUM	API トレースファイル数	－	○	－	○
DCMQCAPITRCSIZE	API トレース容量（エントリ数）	－	○	－	○
DCMQCAPITRCWRITE	API トレース出力単位（エントリ数）	－	○	－	○
DCMQCEXPTRN	トランザクション処理方式の拡張要否	－	○	－	○

（凡例）

○：有効になります。

－：無効になります。

2.2.2 MQC クライアント機能の環境変数のオペランド

MQC クライアント機能を使用するための環境を定義します。

(1) 環境変数のオペランド（共通）

ここでは，UNIX でボーンシェルを使用する場合，または Windows で使用する場合の形式で表記しています。

● DCMQCSRVHOSTNAME = 窓口となる MQC リスナサーバのホスト名
〜＜ 1 〜 255 バイトの文字列＞

窓口となる MQC リスナサーバのホスト名を指定します。
このホスト名は，ドメインネームサーバ（DNS）またはホスト名ファイルに登録されている必要があります。この環境変数には途中に空白文字を含んだ文字列を指定しないでください。途中に空白文字を含んだ文字列を指定した場合，システムログに KFCA30951-E メッセージが出力され，MQC リスナサーバへの接続に失敗します。
DCMQCSRVHOSTIP で窓口となる MQC リスナサーバの IP アドレスを指定した場合は，この指定は省略できます。DCMQCSRVHOSTNAME と DCMQCSRVHOSTIP の両方を指定した場合は，DCMQCSRVHOSTIP の指定が優先されます。

● DCMQCSRVHOSTIP = 窓口となる MQC リスナサーバの IP アドレス
〜（nnn.nnn.nnn.nnn）((0 〜 255))

窓口となる MQC リスナサーバの IP アドレスを指定します。
指定形式は，nnn.nnn.nnn.nnn です。nnn の値をピリオド（.）で区切って指定します。nnn の値は，10 進数で 0 から 255 まで指定できます。また，先頭に 0 を指定した場合は，8 進数値として扱われます。例えば，10 を 010 と指定するとシステムは 8 が指定されたと認識するので注意してください。
DCMQCSRVHOSTNAME で窓口となる MQC リスナサーバのホスト名を指定した場合，この指定は省略できます。DCMQCSRVHOSTNAME と DCMQCSRVHOSTIP の両方を指定した場合は，DCMQCSRVHOSTIP の指定が優先されます。

● DCMQCSRVSERVNAME = 窓口となる MQC リスナサーバのサービス名
〜＜ 1 〜 32 バイトの文字列＞

窓口となる MQC リスナサーバのサービス名を指定します。
このサービス名は，サービス名ファイルに登録されている必要があります。この環境変数には途中に空白文字を含んだ文字列を指定しないでください。途中に空白文字を含んだ文字列を指定した場合，システムログに KFCA30951-E メッセージが出力され，MQC リスナサーバへの接続に失敗します。DCMQCSRVPORT で窓口となる MQC リスナサーバのポート番号を指定した場合は，この指定は省略できます。
DCMQCSRVPORT と DCMQCSRVSERVNAME の両方を指定した場合は，DCMQCSRVPORT の指定が優先されます。
サービス名は，サービス名ファイルに次の形式で指定します。

```
サービス名                ポート番号/TCP
```

● DCMQCSRVPORT = **窓口となる** MQC **リスナサーバのポート番号**

～<符号なし整数> ((5001 ～ 65535))

窓口となる MQC リスナサーバのポート番号を指定します。

DCMQCSRVSERVNAME で窓口となる MQC リスナサーバのサービス名を指定した場合は，この指定は省略できます。DCMQCSRVPORT と DCMQCSRVSERVNAME の両方を指定した場合は，DCMQCSRVPORT の指定が優先されます。

● DCMQCCLTHOSTIP = **自システムの** IP **アドレス**

～（nnn.nnn.nnn.nnn**）**((0 ～ 255))

自システムの IP アドレスを指定します。

指定形式は，nnn.nnn.nnn.nnn です。nnn の値を ".（ピリオド）" で区切って指定します。nnn の値は，10 進数で 0 から 255 まで指定できます。また，先頭に 0 を指定した場合は，8 進数値として扱われます。例えば，10 を 010 と指定するとシステムは 8 が指定されたと認識するので注意してください。

この指定は省略できます。省略した場合は，自システムから自動的に検出された IP アドレスで MQC リスナサーバおよび MQC ゲートウェイサーバに接続します。

● DCMQCCLTPORT = **自システムのポート番号**

～<符号なし整数> ((5001 ～ 65535))

自システムのポート番号を指定します。

OS が自動割り当てするポート番号の範囲外で，かつほかのプログラムが使用しない番号を指定してください。OS が自動割り当てするポート番号は，OS の種別や OS のバージョンなどによって異なります。詳細は，OS のマニュアルを参照してください。

● DCMQCTIMEREQ = MQC **リスナサーバまたは** MQC **ゲートウェイサーバからの結果応答受信監視タイマ値**

～<符号なし整数> ((10 ～ 65535))**《**180**》（単位：秒）**

MQC クライアント機能から MQC リスナサーバ，または MQC ゲートウェイサーバへ要求を送ってから応答が返るまでの待ち時間を指定します。

MQGMO_WAIT オプションを指定した MQGET 命令の場合には，次に示すどちらか小さい方の値が指定時間に加算されます。

- DCMQCTIMEGET で指定した時間
- （MQGET 命令のオプションである WaitInterval フィールドの値）
  －（MQGET 命令発行からの経過時間）

指定時間を過ぎても応答が返らない場合は，API 実行または XA インタフェースがエラーリターンします。

● DCMQCTIMEGET = MQC **ゲートウェイサーバでの** MQGET **命令の待ち合わせ最大時間**

～<符号なし整数> ((1 ～ 60))**《**1**》（単位：秒）**

メッセージを取り出す場合に，MQC ゲートウェイサーバで発行する MQGET 命令の待ち合わせ最大時間を指定します。

MQGET 命令のオプションである WaitInterval フィールドの値が，
DCMQCTIMEGET で指定した値より小さい場合，MQC ゲートウェイサーバは
WaitInterval フィールドの値で MQGET 命令を発行します。
MQGET 命令のオプションである WaitInterval フィールドの値が，
DCMQCTIMEGET で指定した値より大きい場合，MQC ゲートウェイサーバは
DCMQCTIMEGET で指定した値で MQGET 命令を発行します。
時間内にメッセージが取り出せない場合，MQC ゲートウェイサーバは次に示すどちらか小さい方の値で再度，MQGET 命令を発行します。

- DCMQCTIMEGET で指定した時間
- （MQGET 命令のオプションである WaitInterval フィールドの値）－（MQGET 命令発行からの経過時間）

このオペランドは，MQGET 命令で MQGMO_WAIT オプションを指定した場合だけ有効です。

● DCMQCTIMESEG = MQC **リスナサーバまたは** MQC **ゲートウェイサーバからの継続セグメント受信監視タイマ値**

～＜符号なし整数＞ ((10 ～ 65535))《30》**（単位：秒）**

MQC リスナサーバからの応答がセグメント分割される場合に，継続セグメントを受信する時間を指定します。

● DCMQCTIMECON = MQC **リスナサーバまたは** MQC **ゲートウェイサーバへの最大接続試行時間**

～＜符号なし整数＞ ((10 ～ 180))《10》**（単位：秒）**

MQC クライアントが MQC リスナサーバまたは MQC ゲートウェイサーバとの間に TCP/IP コネクションを確立する際に，接続を試行する最大時間を指定します。
MQC クライアントは，TCP/IP コネクションの確立を試行したあと，次に試行するまでの間隔を，1 ミリ秒から始めて順に 2 倍ずつ増やし，最大 1 秒間隔となるまで延ばします。つまり，1 ミリ秒，2 ミリ秒，4 ミリ秒，8 ミリ秒，…，512 ミリ秒，1 秒と間隔を延ばし，以降は 1 秒間隔で試行を続けます。
DCMQCTIMECON に指定する時間は，この TCP/IP コネクションの再試行間隔の合計時間を意味します。TCP/IP コネクションの確立を試行し始めてから，この時間分の再試行を繰り返してもコネクションを確立できなかったとき API はエラーリターンします。
この指定を省略すると，最小値 10 が設定されます。10 未満の値を指定した場合は，設定値は最小値 10 に変更され，180 より大きな値を指定した場合は，設定値は最大値 180 に変更されます。

● DCMQCSEGSIZE = **転送時のセグメントサイズ**

～＜符号なし整数＞ ((1024 ～ 131071))《16384》**（単位：バイト）**

要求または応答をセグメント分割で送受信する場合の転送セグメントサイズを指定します。TCP/IP ウィンドサイズは，転送時のセグメントサイズの 2 倍のサイズで指定されます。

TCP/IP ウィンドウサイズは，このオペランドで指定する転送セグメントサイズを 2 倍した値で取得されます。また，TCP/IP の setsockopt 関数で取得されるウィンドウサイズをサポートしていないシステムでは，API の実行時にエラーリターンする場合があります。この場合は，システムで指定できるウィンドウサイズの範囲を調査の上，このオペランドで指定する転送セグメントサイズを変更してください。

● DCMQCAPITRCOUT = API トレースディスク出力要否

〜 {ON | OFF}《OFF》

API トレース情報の取得を指定します。

- ON：API トレース情報が取得されます。
- OFF：API トレース情報は取得されません。

この指定は省略できます。省略した場合は API トレース情報は取得されません。
API トレース情報を取得するとカレントディレクトリに API トレースファイルが作成されます。ファイル名は次のようになります。

```
mpc.api.ppp.n
```

ppp：プロセス ID
n：0 〜（指定ファイル数 -1）

● DCMQCAPITRCFILNUM = API トレースファイル数

〜＜符号なし整数＞ ((5 〜 65536))《5》

API トレース情報を取得する場合の API トレースファイルの数を指定します。この指定は，DCMQCAPITRCOUT の指定が ON の場合にだけ有効です。
この指定は省略できます。省略した場合は 5 が設定されます。

● DCMQCAPITRCSIZE = API トレース容量（エントリ数）

〜＜符号なし整数＞ ((1024 〜 65536))《1024》

API トレース情報を取得する場合の API トレースファイルの最大エントリ数を指定します。1 エントリは 232 バイトです。このエントリ数を超える API トレース情報の書き込みが発生すると，API トレース情報は折り返します。この指定は，DCMQCAPITRCOUT の指定が ON の場合にだけ有効です。
この指定は省略できます。省略した場合は 1024 が設定されます。

● DCMQCAPITRCWRITE = API トレース出力単位（エントリ数）

〜＜符号なし整数＞ ((0 〜 1024))

UNIX の場合《0》Windows の場合《1》

API トレース情報を取得する場合の API トレースの出力単位を指定します。このエントリ数の API トレース情報の取得，最大エントリ数分の API トレース情報の取得，または MQDISC 命令の発行を契機に，API トレース情報の書き込みが発生します。
この指定は，DCMQCAPITRCOUT の指定が ON の場合にだけ有効です。
API トレースは，MQCONN 命令から MQDISC 命令までが取得されます。また，同一プロセス内で複数回 MQCONN 命令を発行し，それぞれのコネクションハンドルに対して，MQDISC 命令を発行した場合，コネクションハンドルに関係なく，最初に発

行した MQCONN 命令から最後の MQDISC 命令までが取得されます。ただし，同一プロセス内で複数回 MQCONN 命令～ MQDISC 命令を繰り返して発行した場合，API トレースファイルが保持している情報は最後の MQCONN 命令～ MQDISC 命令のトレースです。
この指定は省略できます。省略したときの値は OS の種別によって異なります。Windows では 1 が設定されます。UNIX では 0 が設定され，最大エントリ数分の API トレース情報の取得，または MQDISC 命令の発行を契機に，API トレース情報の書き込みが発生します。

● DCMQCEXPTRN = **トランザクション処理方式の拡張要否**
～ {Y | N}《Y》

トランザクションの子ブランチで XA インタフェースを使用するかどうかを指定します。

- Y：トランザクションの子ブランチで XA インタフェースを使用できます。
- N：トランザクションの子ブランチで XA インタフェースを使用できません。

DCMQCEXPTRN と XA インタフェースの使用の組み合わせによる動作と注意事項を次の表に示します。

表 2-7 DCMQCEXPTRN と XA インタフェースの使用の組み合わせによる動作と注意事項

DCMQCEXPTRN の指定値	XA インタフェースの使用	動作と注意事項
Y	なし	• MQCONN 命令で MQC サーバと接続し，MQDISC 命令で MQC サーバとの接続を解放します。 • 同一スレッド内で複数の MQCONN 命令を発行できます。ただし，同一のコネクションハンドルを複数スレッドで同時に使用した場合の動作は保証できません。 • コネクションハンドルを別スレッドで使用できます。
Y	あり	• XA_OPEN で MQC サーバと接続し，XA_CLOSE で MQC サーバとの接続を解放します。 • トランザクションの子ブランチで XA インタフェースを使用できます。 • ユーザアプリケーションプログラムの同一スレッド内で MQCONN 命令を複数発行すると，2 回目以降の MQCONN 命令は，完了コード MQCC_WARNING，および理由コード MQRC_ALREADY_CONNECTED を返して異常終了します。 • MQCONN 命令はスレッドごとに発行してください。 • MQCONN 命令で取得したコネクションハンドルを異なるスレッドで使用できません。
N	なし	• MQCONN 命令で MQC サーバと接続し，MQDISC 命令で MQC サーバとの接続を解放します。 • 同一スレッド内で複数の MQCONN 命令を発行できます。ただし，同一のコネクションハンドルを複数スレッドで同時に使用した場合の動作は保証できません。 • コネクションハンドルを別スレッドで使用できます。

DCMQCEXPTRN の指定値	XA インタフェースの使用	動作と注意事項
N	あり	• MQCONN 命令で MQC サーバと接続し，MQDISC 命令で MQC サーバとの接続を解放します。 • 同一スレッド内で複数の MQCONN 命令を発行できます。ただし，同一のコネクションハンドルを複数スレッドで同時に使用した場合の動作は保証できません。 • コネクションハンドルを別スレッドで使用できます。 • トランザクションの子ブランチで XA インタフェースを使用できません。

（2） 環境変数のオペランド（JMS インタフェース用）

● ModelQueueName = TemporaryQueue オブジェクト生成時のモデルキュー定義

〜< java.lang.String 型の文字列>

TemporaryQueue（動的キュー）を生成する場合のモデルキュー名を指定します。
48 文字を超える文字列が指定された場合，超えた部分は切り捨てられます。
また，モデルキュー以外のキュー名は指定しないでください。モデルキュー以外のキュー名を指定した場合の動作は保証されません。

● TraceLevel = JMSAPI トレース情報の出力レベル

〜< java.lang.Integer 型の符号なし整数> ((0 〜 4))《3》（単位：トレース出力レベル）

JMSAPI トレース情報の出力レベルを指定します。
出力レベル 5 〜 9 はシステムの障害対策用のレベルのため，通常は使用しないでください。0 〜 9 以外の値を指定すると出力レベル 3（デフォルト）に相当するトレース情報が出力されます。
JMSAPI トレース情報の出力レベルの詳細については，「2.4.3　JMSAPI トレースファイルの出力形式」を参照してください。

● PrfTraceLevel = JMSPRF トレース情報の取得レベル

〜< java.lang.Integer 型の符号なし整数> ((0 〜 2))《1》（単位：トレース取得レベル）

JMSPRF トレース情報の取得レベルを指定します。取得レベル 2 はシステムの障害対策用のレベルのため，通常は使用しないでください。0 〜 2 以外の値を指定すると取得レベル 1（デフォルト）に相当するトレース情報が取得されます。
JMSPRF トレース情報の取得レベルの詳細については，「2.4.4　JMSPRF トレースファイルの取得形式」を参照してください。

● QueueConfigFileName = キュー定義ファイル名

〜< java.lang.String 型の文字列>

キュー定義ファイル名をフルパスで指定します。
このキュー定義ファイルは，JMS インタフェースを使用する場合だけ必要なファイルです。

指定したキュー定義ファイルの読み込みに失敗した場合は，TP1/Message Queue - Access の起動に失敗します。キュー定義ファイルの作成方法については，「2.2.3 キュー定義ファイルの作成（JMS インタフェース用）」を参照してください。

（3） 環境変数のオペランド（接続先情報定義ファイルのキューマネジャ接続用）

● DCMQCDEFCON = **接続先情報定義ファイル読み込み要否**

～ {Y | N}《N》

接続先情報定義ファイルを読み込むかどうかを指定します。

- Y：接続先情報定義ファイルを読み込みます。この指定によって，接続先情報定義ファイルに設定されている複数の MQ サーバに接続できます。ただし，接続先情報定義ファイルに設定できるキューマネジャ構成定義は最大 16 です。それ以上設定しても有効になりません。
  接続先情報定義ファイルが読み込まれるのは，非 XA ライブラリの AP を使用しているときだけです。XA ライブラリ，および JMS インタフェース使用時に設定しても無効となります。
  接続先情報定義ファイルの読み込みを指定した場合は，MQCONN 命令の Name 引数に，環境変数 DCMQCMGRNAME のオペランド値を指定してください。
  MQCONN 命令でキューマネジャ名に空白を指定した場合は，
  MQRC_Q_MGR_NAME_ERR（2058）でエラーになります。
- N：接続先情報定義ファイルを読み込みません。

接続先情報定義ファイルは，該当する接続先情報定義ファイルを使用しているユーザアプリケーションを終了しなくても修正できます。ただし，修正した値を有効にするには，該当する接続先情報定義ファイルを使用しているユーザアプリケーションの停止が必要です。

● DCMQCDEFCONPATH = **接続先情報定義ファイルのパス**

～< 1 ～ 256 バイトの文字列>

接続先情報定義ファイルをフルパスで指定します。パスにはファイル名を含みます。パス名に日本語文字を指定した場合，OS によってシステムログにパス名が正しく表示されません。

2.2.3 キュー定義ファイルの作成（JMS インタフェース用）

JMS インタフェースを使用する場合は，キュー定義ファイルでアプリケーションが使用するキューを定義しておく必要があります。

キュー定義ファイルはテキストエディタを使用して作成します。キュー定義ファイルの名称および格納先ディレクトリは任意です。作成したキュー定義ファイルは，TP1/Message Queue - Access の環境変数 QueueConfigFileName でキュー定義ファイル名を指定することによって有効になります。キュー定義ファイルは TP1/Message Queue - Access の起動時に読み込まれます。キュー定義ファイルの記述形式が不正な場合は，

TP1/Message Queue - Access の起動に失敗します。

(1) キュー定義ファイルの記述形式

```
QueueImplClass=QImpl名※1
Queue.通番※2.DisplayName=キューの表示名※3
Queue.通番.QueueName=キュー名称※4
Queue.通番.QueueManagerName=キューマネジャ名称※5
```

注※ 1

QImpl 名は「jp.co.Hitachi.soft.mqadaptor.QueueImpl」固定です。

注※ 2

通番には，1 ～ 20480 の範囲の値を指定してください。

注※ 3

Cosminexus の NameSpace には，「TP1_Message_Queue_Access_ 表示名 _que」の値で登録されます。

注※ 4

キュー名称には，モデルキュー以外のキューを指定してください。

注※ 5

キューマネジャ名称には，接続先のキューマネジャ名称を指定してください。

(2) キュー定義ファイルの記述例

```
QueueImplClass=jp.co.Hitachi.soft.mqadaptor.QueueImpl
Queue.1.DisplayName=Que1
Queue.1.QueueName=Queue1
Queue.1.QueueManagerName=QMgr1
Queue.2.DisplayName=Que2
Queue.2.QueueName=Queue2
Queue.2.QueueManagerName=QMgr2
Queue.3.DisplayName=Que3
Queue.3.QueueName=Queue3
Queue.3.QueueManagerName=QMgr3
```

(3) キュー定義ファイルの作成規則

- コメントや継続行の記述も含めて，java.util.Properties クラスの load メソッドで読み出しできる形式で作成してください。
- QueueImplClass の指定がない場合，または QueueImplClass の値が指定されていない場合はデプロイがエラーになります。
- 個々のキューごとに，DisplayName，QueueName，および QueueManagerName のそれぞれを指定してください。
- Queue.n.DisplayName の指定がない場合，または Queue.n.DisplayName の値が指定されていない場合は，通番が n-1 のキューまでが有効になります。この場合，通番

n 以降のキュー定義は無視されます。

- QueueName の指定がない，または QueueName の値が指定されていないキューは無効になります。
- QueueManagerName の指定がない，QueueManagerName の値が指定されていない，またはすべて空白が指定されているキューは，デフォルトのキューマネジャが指定されたものと認識されます。
- 同一のプロパティを複数記述した場合は，どの値が有効になるか保証されません。
- キュー定義が一つもない場合は，フォーマット不正となります。
- 表示名が重複した場合，通番の大きい方が有効となります。
- DisplayName に指定できる文字は，半角英字（A ～ Z，a ～ z），半角数字（0 ～ 9），および半角のアンダスコア（_）です。それ以外の文字を指定したキューは無効になります。
- キュー名称には，モデルキューは指定しないでください。モデルキューを指定した場合，該当するモデルキューを指定した createSender，createReceiver，および createBrowser，ならびに Queue 指定の send が発行されると，InvalidDestinationException がスローされます。

2.2.4 XA インタフェースの設定項目

MQC クライアント機能を使用する場合，XA インタフェースの設定が必要です。設定方法の詳細については，使用する TM のマニュアルを参照してください。

(1) 設定項目

- RM 名

  `MQClient`

- RM スイッチ名

  `dc_xa_mqc_switch`

- RM 関連オブジェクトファイル名

  適用 OS に応じて次の表に示す，RM 関連オブジェクトファイル名，およびリンケージオプションを設定してください。

表 2-8 RM 関連オブジェクトファイル名

適用 OS	RM 関連 オブジェクトファイル名	リンケージオプション
AIX	`libmqcx.a`	`-brtl -lpthread -lC`
HP-UX	`libmqcxp.sl`	`-lpthread`
Linux	`libmqcx.so`	`-lpthread`

適用 OS	RM 関連 オブジェクトファイル名	リンケージオプション
Windows	libmqcx.lib	―

（凡例）

―：該当しません。

（2）XA インタフェースのバージョン互換

XA インタフェースのバージョン互換について説明します。

● 1 スレッド内で複数の MQCONN 命令を発行する場合

- TP1/Message Queue - Access 05-00 未満，または TP1/Message Queue - Access 05-00 以降で環境変数に DCMQCEXPTRN=N を設定したとき
  MQCONN 命令は正常終了します（MQCC_OK，MQRC_NONE）。
- TP1/Message Queue - Access 05-00 以降で環境変数に DCMQCEXPTRN=Y を設定したとき
  2 回目以降の MQCONN 命令は異常終了します（MQCC_WARNING，MQRC_ALREADY_CONNECTED）。

● トランザクションを使用する場合

- TP1/Message Queue - Access 05-00 未満，または TP1/Message Queue - Access 05-00 以降で環境変数に DCMQCEXPTRN=N を設定したとき
  子ブランチでは使用できません。
- TP1/Message Queue - Access 05-00 以降で環境変数に DCMQCEXPTRN=Y を設定したとき
  子ブランチで使用できます。

● MQC サーバと接続する場合

- TP1/Message Queue - Access 05-00 未満，または TP1/Message Queue - Access 05-00 以降で環境変数に DCMQCEXPTRN=N を設定したとき
  MQCONN 命令で MQC サーバと接続し，MQDISC 命令で MQC サーバとの接続を解放します。
- TP1/Message Queue - Access 05-00 以降で環境変数に DCMQCEXPTRN=Y を設定したとき
  XA_OPEN で MQC サーバと接続し，XA_CLOSE で MQC サーバとの接続を解放します。

（3）XA インタフェースを使用する場合の注意事項

● TM と接続する場合，xa_open 関数を発行したあとで MQI，または C++ インタフェースの MQI に相当するメソッドを発行してください。

● xa_open 関数はスレッド単位で発行してください。

● TP1/Message Queue - Access は動的結合する RM です。

2.2.5 接続先情報定義ファイルの設定項目

接続先情報定義ファイルには，MQC クライアント機能を使用するための環境を定義します。

(1) 環境変数のオペランド（接続先情報定義ファイル）

接続先情報定義ファイル独自のオペランドを次に示します。そのほかのオペランドについては，「2.2.2 MQC クライアント機能の環境変数のオペランド」を参照してください。有効になる環境変数の設定場所については，「表 2-6 接続先情報定義ファイルの読み込みの有無による環境変数の設定場所の有効性」を参照してください。

● DCMQCMGRNAME = **接続先の** MQ **サーバのキューマネジャ名**
～< 1 **～** 48 **バイトの** MQ **文字列>**

接続先の MQ サーバのキューマネジャ名を指定します。
このキューマネジャ名は，接続先の MQA サービス定義の mqaquemgr -n オプションで指定されているキューマネジャ名を指定してください。同一キューマネジャ名を指定し，IP アドレスやポート番号が異なる場合は，定義ファイルで最初に設定したキューマネジャの値が有効になります。

(2) 接続先情報定義ファイルの記述形式

```
MGR
DCMQCMGRNAME=キューマネジャ名※1
DCMQCSRVHOSTIP=窓口となるMQCリスナサーバのIPアドレス※2
DCMQCSRVHOSTNAME=窓口となるMQCリスナサーバのホスト名※2
DCMQCSRVPORT=窓口となるMQCリスナサーバのポート番号※3
DCMQCSRVSERVNAME=窓口となるMQCリスナサーバのサービス名※3
DCMQCCLTHOSTIP=自システムのIPアドレス
DCMQCCLTPORT=自システムのポート番号
DCMQCTIMEREQ=MQCリスナサーバまたはMQCゲートウェイサーバからの結果応答
             受信監視タイマ値
DCMQCTIMEGET=MQCゲートウェイサーバでのMQGET命令の待ち合わせ最大時間
DCMQCTIMESEG=MQCリスナサーバまたはMQCゲートウェイサーバからの継続
             セグメント受信監視タイマ値
DCMQCTIMECON=MQCリスナサーバまたはMQCゲートウェイサーバへの最大接続試行
             時間
DCMQCSEGSIZE=転送時のセグメントサイズ
MGREND
```

注※ 1

必須環境変数です。定義値に空白を指定しないでください。

注※ 2

必須環境変数です。どちらか一つを指定してください。両方指定した場合は DCMQCSRVHOSTIP が有効になります。

注※ 3

必須環境変数です。どちらか一つを指定してください。両方指定した場合は DCMQCSRVPORT が有効になります。

（3） 接続先情報定義ファイルの作成規則

1. 「MGR」から「MGREND」まで（以降，キューマネジャ構成定義と呼びます）が一つの接続先の情報となるため，この間に各環境変数を設定してください。それ以外の場所に環境変数を設定した場合，その行は無効になります。また，「MGR」を設定したあと，「MGREND」を設定しないで，再び「MGR」を設定した場合は，その「MGR」行は無効になります。
2. 接続先情報定義ファイルに設定しているキューマネジャ構成定義は，最大 16 個まで読み込まれます。
3. 1 行に 301 バイト以上の文字列を指定したとき，301 バイト以降は無効になります（改行は文字としてカウントされません）。
4. コメントを入れる場合は，コメントの先頭に「#」を指定してください。
5. 接続先情報定義ファイルに指定する文字は，半角で記述してください。全角で記述した場合，次のどれかになります。
   - 無効な行になります。
   - 変数名が指定されていないと判断されます。
   - 定義値に不正な値が設定されていると判断されます。

   ただし，ホスト名，サービス名の定義値，または「#」以降のコメント文字は，全角で記述できます。
6. キューマネジャ構成定義内で，同じ環境変数を指定した場合，あとから指定した値が有効となります。
7. 複数のキューマネジャ構成定義が設定されているとき，環境変数 DCMQCMGRNAME に同一キューマネジャ名を指定し，ほかの環境変数（DCMQCSRVHOSTIP など）が異なる場合は，最初に設定されているキューマネジャ構成定義だけが有効になります。
8. 環境変数は 1 行に一つだけ指定できます。複数指定した場合，二つ目以降の環境変数は無効となります。
9. 接続先情報定義ファイルのオペランドにない環境変数を指定した場合は，その行は無効になります。
10. 「=」より前を環境変数，「=」より後ろを定義値として扱います。「=」を同じ行に二つ以上指定しても二つ目以降の「=」は無効となります。
11. 環境変数の後ろに，空白文字，またはタブを指定した場合は，そこから「=」または改行の前までが無効となります。また，定義値の後ろに空白文字，タブ，「=」，または「#」を指定した場合，該当する行の，空白文字，タブ，「=」，または「#」以降が

無効になります。

12.「MGR」，および「MGREND」の前に設定できるのは，空白文字とタブだけです。後ろに設定できるのは空白文字とタブ，またはコメントです。それ以外の文字を指定した場合，その行は無効になります。

13.キューマネジャ構成定義内で必須環境変数を省略した場合，または，環境変数の定義値に何も指定していない場合は，該当するキューマネジャ構成定義は無効になります。

14.接続先情報定義ファイルに設定できる最大行数は 720 です。それ以上は設定しても無効になります。

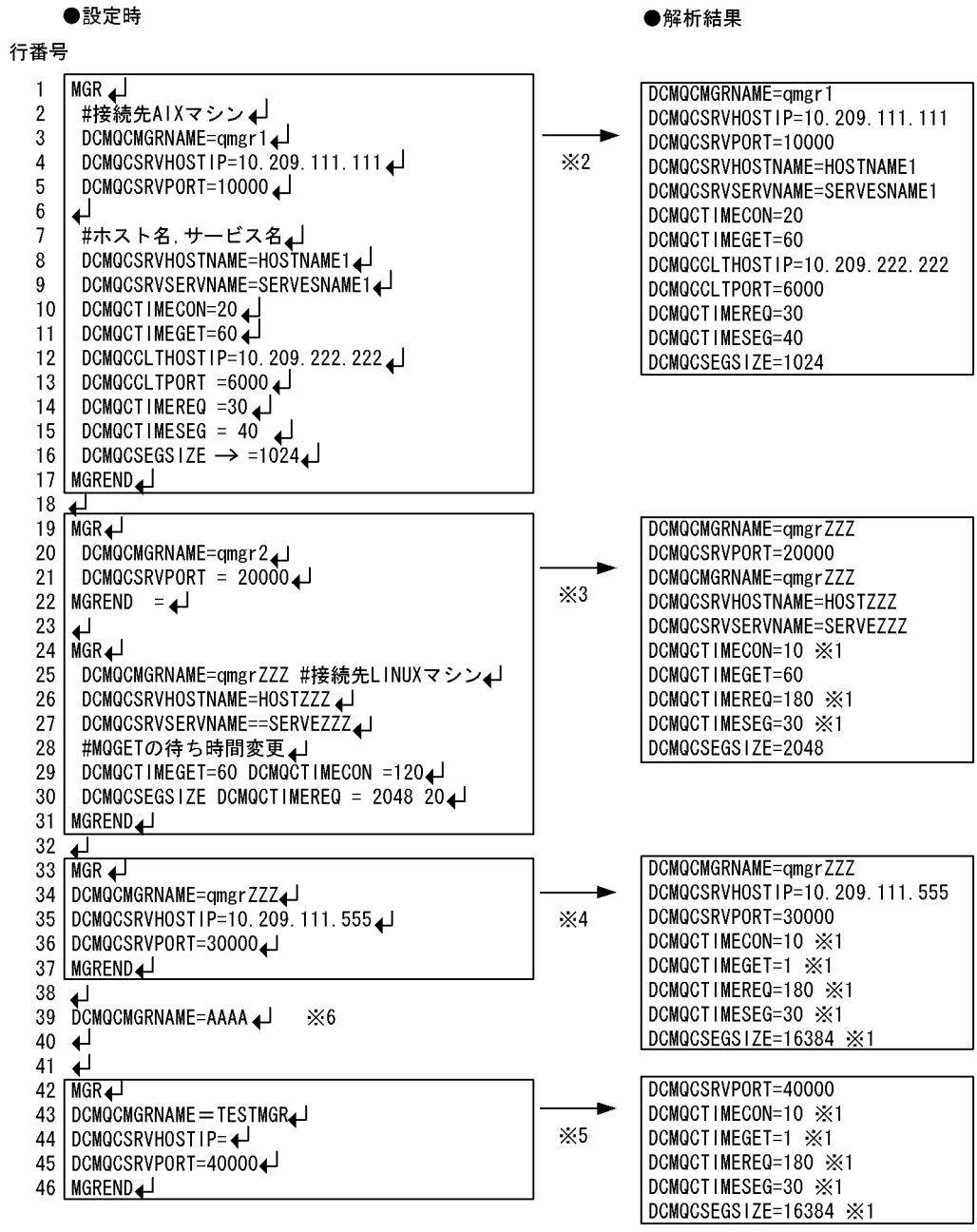
15.接続先情報定義ファイルで，次のような設定をした場合，フォーマット不正として，MQCONN で MQRC_UNEXPECTED_ERROR（2195）のエラーとなります。

- キューマネジャ構成定義を一つも定義していない場合
- 接続先情報定義ファイルに設定している，すべてのキューマネジャ構成定義が無効な場合

（4） 接続先情報定義ファイルの記述例

接続先情報定義ファイルの設定と解析結果を次の図に示します。

図 2-1　接続先情報定義ファイルの設定と解析結果



（凡例）
↵：改行を示します。
→：タブを示します。

注※ 1
デフォルト値であることを示します。

注※ 2

環境変数はすべて正しく設定されているため，すべて有効な定義値として読み込まれます。ただし，環境変数 DCMQCSRVHOSTNAME および DCMQCSRVSERVNAME は，それぞれ DCMQCSRVHOSTIP および DCMQCSRVPORT が設定されて有効になっているため，無効な値となります。

注※ 3

20 行目で環境変数 DCMQCMGRNAME を設定していますが，25 行目でも DCMQCMGRNAME を設定しているため，作成規則 6. によって 25 行目の DCMQCMGRNAME で設定した値が有効になります。
22 行目は作成規則 12. によって無効な行となります。そのため，作成規則 1. によって 24 行目も無効な行となり，19 行目から 31 行目までが一つのキューマネジャ構成定義となります。
29 行目は作成規則 8. によって，先に設定されている環境変数 DCMQCTIMEGET=60 が有効になり，あとに設定した環境変数 DCMQCTIMECON=120 は無効となります。
30 行目は作成規則 11. によって，環境変数 DCMQCSEGSIZE の後ろに空白文字が指定されているため，環境変数 DCMQCTIMEREQ が無効になります。同様に定義値の「2048」の後ろに空白文字が指定されているため，「20」は無効となります。

注※ 4

33 行目から 37 行目のキューマネジャ構成定義は，34 ～ 36 行目の必須環境変数は正しく設定されており，ほかの環境変数にデフォルトの値が設定されています。しかし，19 行目からのキューマネジャ構成定義と同じキューマネジャ名を設定しているため，作成規則 7. によって，このキューマネジャ構成定義は無効となります。

注※ 5

43 行目の「DCMQCMGRNAME ＝ TESTMGR」で「＝」は全角であるため，作成規則 5. および 10. によって，この 1 行が環境変数として解析されます。そのため，必須環境変数である，「DCMQCMGRNAME」は設定されていないと判断されます。また，44 行目で定義値を設定していないため，作成規則 13. によって，このキューマネジャ構成定義は無効になります。

注※ 6

39 行目は，作成規則 1. によって，キューマネジャ構成定義外に設定しているため，この行は無効となります。

2.3 クライアントアプリケーションの作成

MQC クライアント機能のクライアントアプリケーションの作成時に使用できる言語，リンケージオプション，およびトランザクションに関する注意事項について説明します。

2.3.1 使用できる言語

MQC クライアント機能のクライアントアプリケーションの作成時には，次に示すプログラミング言語を使用できます。

● C 言語

C 言語の MQI を使用して作成します。詳細については，「4. MQC クライアント機能の MQI」を参照してください。

C 言語で作成したクライアントアプリケーションで必要なリンケージオプションを次の表に示します。

表 2-9 リンケージオプション（C 言語）

適用 OS	トランザクション連携の種別	リンケージオプション
AIX	XA インタフェース接続	`-lmqcx -brtl -lpthread -lC`
	非 XA インタフェース接続	`-lmqc -brtl -lpthread -lC`
HP-UX，Linux	XA インタフェース接続	`-lmqcx -lpthread`
	非 XA インタフェース接続	`-lmqc -lpthread`
Windows	XA インタフェース接続	`libmqcx.lib`
	非 XA インタフェース接続	`libmqc.lib`

● COBOL 言語

COBOL 言語の MQI を使用して作成します。詳細については，「4. MQC クライアント機能の MQI」を参照してください。

COBOL 言語で作成したクライアントアプリケーションで必要なリンケージオプションを次の表に示します。

表 2-10 リンケージオプション（COBOL 言語）

適用 OS	トランザクション連携の種別	リンケージオプション
AIX	XA インタフェース接続	`-lmqccbx -brtl -lpthread -lC`
	非 XA インタフェース接続	`-lmqccb -brtl -lpthread -lC`
HP-UX，Linux	XA インタフェース接続	`-lmqccbx -lpthread`
	非 XA インタフェース接続	`-lmqccb -lpthread`

適用 OS	トランザクション連携の種別	リンケージオプション
Windows	XA インタフェース接続	libmqccbx.lib
	非 XA インタフェース接続	libmqccb.lib

● C++ 言語

C++ 言語を使用して作成します。詳細については，「5. MQC クライアント機能の C++ インタフェース」を参照してください。

C++ 言語で作成したクライアントアプリケーションで必要なリンケージオプションを次の表に示します。

表 2-11 リンケージオプション（C++ 言語）

適用 OS	トランザクション連携の種別	リンケージオプション
AIX	XA インタフェース接続	-brtl -lmqccpp -lmqcx -lpthread
	非 XA インタフェース接続	-brtl -lmqccpp -lmqc -lpthread
HP-UX，Linux	XA インタフェース接続	-lmqccpp -lmqcx -lpthread
	非 XA インタフェース接続	-lmqccpp -lmqc -lpthread
Windows	XA インタフェース接続	libmqcppx.lib
	非 XA インタフェース接続	libmqcpp.lib

● Java 言語

Java 言語を使用して作成します。詳細については，「6. MQC クライアント機能の Java インタフェース」を参照してください。

JTA インタフェース接続をする場合は，Java 言語の JMS インタフェースを使用して作成します。詳細については，「7. MQC クライアント機能の JMS インタフェース」を参照してください。

TP1/Message Queue - Access の JMS インタフェースは，Sun Microsystems, Inc. が提供する JMS 1.0 のインタフェースに基づいて実装しています。JMS 1.0 と MQC クライアント機能の JMS インタフェースとの機能差については，「付録 B JMS 仕様と MQC クライアント機能の JMS インタフェースとの差異」を参照してください。

2.3.2 注意事項

クライアントアプリケーションの作成，および実行時の注意事項について次に示します。

● TP1/Message Queue - Access は 2 相コミットに対応しています。ただし，ローカルトランザクション機能は 1 相コミットです。

● XA インタフェースについて，次に示す機能は使用できません。
- トランザクションブランチとスレッドの結合の移行（マイグレーション）

- XA 関数の非同期動作

また，xa_open 関数はスレッド単位で発行してください。

● MQC クライアント機能は，トランザクションの決着を MQC サーバ機能に依頼します。そのため，回線障害やサーバダウンが原因でトランザクションの決着時に MQC サーバ機能と連絡が取れない場合は，TM の決着依頼に対しリトライを要求します。したがって，トランザクション中に MQC サーバ機能を停止した場合は，MQC クライアント機能は MQC サーバ機能が再起動するまで，TM の決着依頼に対し繰り返しリトライを要求します。

● TP1/Message Queue - Access は，次の命令のエラー時に理由コードとして MQRC_CONNECTION_BROKEN（コネクション切断）を返す場合があります。この理由コードを受け取った場合は，MQI に指定したコネクションハンドルが無効になります。MQCONN 命令の発行によってコネクションを再度設定してください。
- MQDISC，MQOPEN，MQCLOSE，MQGET，MQPUT，MQPUT1，MQSET，MQINQ，MQCMIT，または MQBACK

● TP1/Message Queue - Access を使用する UAP で，MQGMO 構造体の Options パラメタに MQGMO_WAIT を指定した場合は，MQGET 命令が完了するまで，UAP を正常に停止させることはできません。

● UAP を Java で作成する場合，Java がプロセスメモリを解放します。このため，メモリ不足（理由コード：SIGSEGV）で Java が異常終了する場合があります。Java 起動時に，-mxXXm（XX：サイズ）を指定して最大ヒープサイズを調整してください。

● TP1/Message Queue - Access を XA 接続して使用する場合，トランザクション発生（トランザクションマネージャが提供する関数発行）後のトランザクション内指定の MQPUT 命令，MQPUT1 命令，または MQGET 命令を発行したときに，サーバでトランザクションが発生します。
その後，トランザクションが決着するまでの間，トランザクションを発生させたクライアントのスレッドで発行された MQCONN 命令は，理由コードとして MQRC_ALREADY_CONNECTED（アプリケーションはすでに接続されています）を返し，そのトランザクションで使用しているコネクションハンドルを返します。

● TP1/Message Queue - Access は，MQDISC 命令発行時に MQC ゲートウェイサーバで後処理をするため，直後に発行された MQCONN 命令が MQRC_Q_MGR_NOT_AVAILABLE（キューマネジャが接続不能）でエラーリターンする場合があります。
MQDISC 命令，または MQCONN 命令を繰り返し発行する UAP を使用した場合は，同時 MQCONN 数と MQC ゲートウェイサーバ数が同じであるときにも，MQCONN 命令のリトライ発行処理をしてください。

● TP1/Message Queue - Access を使用する UAP では，MQCONN 命令発行時に回線障害（ソケット関数異常終了）が発生した場合に，理由コードとして

MQRC_UNEXPECTED_ERROR（予期しないエラーが発生）を返します。また，MQC ゲートウェイサーバが不足した場合には，理由コードとしてMQRC_Q_MGR_NOT_AVAILABLE（キューマネジャが接続不能）を返します。どちらの場合も UAP で MQCONN 命令のリトライ発行処理をしてください。

- サーバ側に対するトランザクション制御依頼の送受信中にクライアントアプリケーションがダウンした場合，サーバ側のトランザクション状態とクライアント側のトランザクション状態が不一致となることがあります。この場合，サーバ側で次の対処をすることでトランザクションをロールバックできます。
  1. OpenTP1 のトランザクション状態表示コマンド（trnls -t）を実行し，不一致となったトランザクションを発生させたサーバ名（MQCGWP サーバ名）を確認します。
  2. 確認した MQCGWP サーバを dcsvstop コマンドで正常停止します。
  3. 1. で確認した MQCGWP サーバに対して mqccontrn コマンドに -f オプションを指定して実行します。
- XA 連携を行う場合，コネクションハンドルとトランザクションは 1 対 1 の関係で使用してください。一つのコネクションハンドルを複数のトランザクションで同時に使用できません。また，トランザクション内で複数のコネクションハンドルを使用できません。
- JMS インタフェースの API トレース取得情報について
  JMS インタフェースの API トレースは，MQI レベルの情報が取得されます。JMS インタフェースでは，デプロイ以降で発行されたすべての MQI 情報が取得されます。

2.4 MQC クライアント機能の障害対策

MQC クライアント機能が障害時に取得する情報について次の表に示します。

なお，次の表に示す障害情報を残しておきたい場合は，必要に応じてコピーしてください。

表 2-12 MQC クライアント機能が障害時に取得する情報

取得情報	出力情報	参照方法
コアファイル（.core）※1	内部情報のプロセスなど	－
API トレースファイル（mqc.api.XXX.ZZ ※2）	API 情報など	mqcapiout コマンドで取得して参照してください。
Java コンソール	Java のメソッド情報など	－
JavaEnvironment トレース	Java のメソッド情報など	－
JMSAPI トレースファイル（*Cosminexus インストールディレクトリ*/CC/server/public/ejb/ *サーバ名称*/logs/connectors/TP1_Message_Queue_Access*.log ※3）	JMS インタフェースのメソッド情報など	－
JMSPRF トレースファイル（*スプールディレクトリ (prfspool)*/utt/prf/*PRF 識別子*/dcopltrc/prf_*）	JMS インタフェースの性能検証用情報	Cosminexus Performance Tracer の cprfflush コマンドで取得し，cprfed コマンドで編集して参照してください
システムログファイル	TP1/Message Queue - Access が出力するメッセージ	－

（凡例）

－：該当しません。

注※ 1

OS が Windows の場合，取得できません。

注※ 2

XXX：プロセス ID

ZZ：0 ～ 65536

注※ 3

*斜体*で示す部分はシステム構成によって値が変わる部分です。また，Windows の場合，ディレクトリ名の区切りは "/" ではなく "¥" になります。

2.4.1 API トレースファイルの出力形式

API トレースファイルは，mqcapiout コマンドで取得できます。出力形式については，3 章の「mqcapiout（API トレースファイル取得）」を参照してください。

2.4.2 JavaEnvironment トレースファイルの出力形式

JavaEnvironment トレースファイルには，Java のメソッドに関する情報が出力されます。

ここでは，JavaEnvironment トレース情報の出力形式について説明します。〔 〕で囲んだ項目は，出力レベルの指定値によって出力の有無を選択できる項目です。

なお，JavaEnvironment トレース情報の出力レベルは，MQEnvironment クラスの enableTracing メソッドで指定します。出力レベルの詳細については，6 章の「MQEnvironment クラス（Java）」を参照してください。

(1) 出力形式

```
XX....XX YY....YY
〔Value = AA....AA〕
〔送信／受信のMQヘッダ〕
〔MQGMO情報 | MQPMO情報〕
〔送信／受信ユーザメッセージデータ〕
```

(2) 出力項目の説明

XX....XX

記入項目（文字列）を「記入項目 = クラス名 @ ハッシュ値 # メソッド名」の形式で出力します。

YY....YY

発行個所（文字列）を次の値で出力します。

- Start：入口
- End：出口

AA....AA

パラメタ情報（文字列）です。

(3) 送信／受信の MQ ヘッダ

● 送信／受信の MQ ヘッダの出力形式

```
*** MQMessage Variable *** @ aa....aa
report = bb....bb
messageType = cc....cc
expiry = dd....dd
feedback = ee....ee
encoding = ff....ff
characterSet = gg....gg
```

```
format = hh....hh
priority = ii....ii
persistence = jj....jj
messageId = kk....kk
correlationId = ll....ll
backoutCount = mm....mm
replyToQueueName = nn....nn
replyToQueueManagerName = oo....oo
userId = pp....pp
accountingToken = qq....qq
applicationIdData = rr....rr
putApplicationType = ss....ss
putApplicationName = tt....tt
putDateTime = uu....uu
applicationOriginData = vv....vv
groupId = ww....ww
messageSequenceNumber = xx....xx
offset = yy....yy
messageFlags = zz....zz
originalLength = AA....AA
```

● 送信／受信の MQ ヘッダの出力項目の説明

`aa....aa`
　ハッシュコード（16 進数）です。

`bb....bb`
　報告オプション（10 進数）です。

`cc....cc`
　メッセージタイプ（10 進数）です。

`dd....dd`
　メッセージ保持時間（10 進数）です。

`ee....ee`
　返答コード（10 進数）です。

`ff.....ff`
　マシンコード形式（10 進数）です。

`gg....gg`
　文字セット識別子（10 進数）です。

`hh....hh`
　フォーマット名（文字列）です。

`ii....ii`
　メッセージ優先度（10 進数）です。

`jj....jj`
　メッセージ永続性（10 進数）です。

`kk....kk`
　メッセージ識別子の内容（16 進数）です。

ll....ll
　相関識別子（16 進数）です。

mm....mm
　ロールバックカウンタ（10 進数）です。

nn....nn
　応答キュー名（文字列）です。

oo....oo
　応答キューマネジャ名（文字列）です。

pp....pp
　ユーザ識別子（文字列）です。

qq....qq
　課金トークン（16 進数）です。

rr....rr
　アプリケーション識別データ（文字列）です。

ss....ss
　登録アプリケーションタイプ（10 進数）です。

tt....tt
　登録アプリケーション名（文字列）です。

uu....uu
　登録日付および登録時刻（文字列）です。

vv....vv
　登録元データ（文字列）です。

ww....ww
　グループ識別子（16 進数）です。

xx....xx
　メッセージシーケンス番号（10 進数）です。

yy....yy
　オフセット（10 進数）です。

zz....zz
　メッセージフラグ（10 進数）です。

AA....AA
　登録元メッセージ長（10 進数）です。

（4） MQGMO 情報

● MQGMO 情報の出力形式

```
*** MQGetMessageOptions Variable *** @ aa....aa
```

```
options = bb....bb
waitInterval = cc....cc
resolvedQueueName = dd....dd
matchOptions = ee....ee
groupStatus = ff....ff
segmentStatus = gg....gg
segmentation = hh....hh
```

● MQGMO 情報の出力項目の説明

aa....aa
　ハッシュコード（16 進数）です。

bb....bb
　取り出しオプション（10 進数）です。

cc....cc
　待ち合わせ最大時間（10 進数）です。

dd....dd
　受信キュー名（文字列）です。

ee....ee
　一致オプション（10 進数）です。

ff.....ff
　メッセージグループフラグ（文字列）です。

gg....gg
　論理メッセージフラグ（文字列）です。

hh....hh
　セグメント分割フラグ（文字列）です。

（5） MQPMO 情報

● MQPMO 情報の出力形式

```
*** MQPutMessageOptions Variable *** @ aa....aa
options = bb....bb
resolvedQueueName= cc....cc
resolvedQueueManagerName= dd....dd
knownDestCount= ee....ee
unknownDestCount= ff....ff
invalidDestCount= gg....gg
contextReference= hh....hh
recordFields = ii....ii
```

● MQPMO 情報の出力項目の説明

aa....aa
　ハッシュコード（16 進数）です。

bb....bb
登録オプション（10 進数）です。

cc....cc
受信キュー名（文字列）です。

dd....dd
受信キューマネジャ名（文字列）です。

ee....ee
ローカルキューの数（10 進数）です。

ff.....ff
リモートキューの数（10 進数）です。

gg....gg
送信に失敗したキューの数（10 進数）です。

hh....hh
コンテキスト（文字列）です。

ii....ii
MQPMR 構造体フィールドの存在（10 進数）です。

（6）送信／受信ユーザメッセージデータ

● 送信／受信ユーザメッセージデータの出力形式

```
aa....aa message size: bb....bb
aa....aa message: cc....cc
```

● 送信／受信ユーザメッセージデータの出力項目の説明

aa....aa
メッセージの種類（文字列）を次の値で出力します。
・Get：Get メッセージ
・Put：Put メッセージ

bb....bb
メッセージデータのサイズ（10 進数）です。

cc....cc
メッセージデータ（16 進数）です。

2.4.3 JMSAPI トレースファイルの出力形式

JMSAPI トレースファイルには，ユーザ API および Cosminexus インタフェースのメソッドに関する情報が出力されます。

ここでは，JMSAPI トレース情報の出力レベルと出力形式について説明します。

(1) 出力レベル

JMSAPI トレースファイルでは，次の表に示す 5 とおりの出力レベルを指定できます。デフォルトは「レベル 3」です。

表 2-13 JMSAPI トレース情報の出力レベル

ユーザ API および Cosminexus インタフェースのメソッドに関する出力情報	出力レベル				
	レベル 0 ※	レベル 1	レベル 2	レベル 3	レベル 4
入口情報	×	○	○	○	○
正常時の出口情報	×	○	○	○	○
異常時の出口情報（Message インタフェース以外）	×	○	○	○	○
送信／受信メッセージデータ	×	×	○	○	○
パラメタなどの埋め込み情報	×	×	×	○	○
Message インタフェースのトレース情報	×	×	×	×	○

（凡例）

○：出力されます。

×：出力されません。

注※

レベル 0 では，ユーザ API および Cosminexus インタフェースのメソッドに関する情報は出力されません。KFCA から始まるメッセージ（TP1/Message Queue - Access のメッセージ），および例外スタックトレースだけが出力されます。

(2) 出力形式

各トレース情報の出力形式を示します。〔 〕で囲んだ項目は，出力レベルの指定値によって出力の有無を選択できる項目です。

(a) 入口情報の出力形式

```
(AA....AA)yyyy-mm-dd hh:mi:ss.mss          〔Hconn = BB....BB〕
 〔Xid = [0x CC....CC] [0x CC....CC]〕
XX....XX # YY....YY START
 〔QueueName = DD....DD〕
 〔Value1 = EE....EE
  Value2 = EE....EE
      :
      :
  ］
 〔MessageObject = FF....FF〕
 〔送信／受信メッセージデータ〕
```

(b) 正常時の出口情報の出力形式

```
(AA....AA)yyyy-mm-dd hh:mi:ss.mss          〔Hconn = BB....BB〕
 〔Xid = [0x CC....CC] [0x CC....CC]〕
XX....XX # YY....YY END
 〔QueueName = DD....DD〕
 〔Value1 = EE....EE
  Value2 = EE....EE
      :
      :
  ]
 〔MessageObject = FF....FF〕
 〔送信／受信メッセージデータ〕
```

(c) 異常時の出口情報の出力形式

```
(AA....AA)yyyy-mm-dd hh:mi:ss.mss          〔Hconn = BB....BB〕
 〔Xid = [0x CC....CC] [0x CC....CC]〕
XX....XX # YY....YY ERR
 〔QueueName = DD..DD〕
 〔Value1 = EE....EE
  Value2 = EE....EE
      :
      :
  ]
 〔MessageObject = FF....FF〕
 〔CompletionCode = GG....GG〕
 〔ReasonCode = HH....HH〕
```

(d) 出力項目の説明

AA....AA

トレースタイプを次の値で出力します。

- API：API トレースレベル
- COS：Cosminexus インタフェースレベル

yyyy-mm-dd

取得した年（西暦）月日（10 進数）です。

hh:mi:ss.mss

取得した時分秒ミリ秒（10 進数）です。

BB....BB

コネクションハンドルの値（10 進数）です。

CC....CC

トランザクション ID（16 進数）です。

XX....XX

発行元オブジェクトのクラス名称（文字列）です。

YY....YY

発行元のメソッド名称（文字列）です。

DD....DD
キュー名称（文字列）が埋め込み情報として出力されます。

EE....EE
パラメタのなどの埋め込み情報（int 型，String 型，long 型，float 型，double 型，boolean 型，short 型，char 型，byte 型，または Object 型）です。

FF....FF
Message オブジェクトのクラス名称およびハッシュコード（文字列）が埋め込み情報として出力されます。

GG....GG
完了コード（10 進数）です。

HH....HH
理由コード（10 進数）です。

(e) 送信／受信メッセージデータ

● 送信／受信メッセージデータの出力形式

```
<<<----- Message Data ----->>>
message size = aa....aa
message =
bb....bb
[0x cc....cc]
```

● 送信／受信メッセージデータの出力項目の説明

aa....aa
メッセージデータのサイズ（10 進数）です。

bb....bb
メッセージデータ（文字列）です。最大 64 バイトを出力します。
メッセージデータの中に，MQ 文字列でも空白でもない文字が含まれている場合は，その文字をピリオド（.）に変換して出力します。

cc....cc
メッセージデータ（16 進数）です。最大 64 バイトを出力します。

2.4.4 JMSPRF トレースファイルの取得形式

JMSPRF トレースファイルには，性能検証用の情報が出力されます。

JMSPRF トレースを編集するには，Cosminexus Performance Tracer をインストールする必要があります。また，トレース情報の強制ファイル出力（cprfflush コマンド），およびトレース情報ファイルの編集出力（cprfed コマンド）については，マニュアル「Cosminexus リファレンス コマンド編」を参照してください。

ここでは，JMSPRF トレースの取得レベルとイベント ID ごとの取得内容について説明します。

(1) 取得レベル

JMSPRF トレースファイルでは取得レベルを指定できます。デフォルトは「レベル 1」です。

- 0：JMSPRF トレースは取得しません。
- 1：イベント ID が 0x920，0x921，0x92A，0x92B で始まる情報を取得します。

(2) 取得ポイント

トレース情報が取得されたタイミングは，出力されたイベント ID で識別します。イベント ID および TP1/Message Queue - Access の JMSPRF トレース情報の取得ポイントを次の表に示します。

表 2-14 JMSPRF トレース情報の取得ポイント

イベント ID	トレース取得ポイント	詳細情報データ長（単位：バイト）
0x9200	send(message only) の入口	0
0x9210	send(message only) の出口	152
0x9201	send(message and 3 parameters) の入口	0
0x9211	send(message and 3 parameters) の出口	152
0x9202	send(queue, message) の入口	0
0x9212	send(queue, message) の出口	152
0x9203	send(queue, message and 3 parameters) の入口	0
0x9213	send(queue, message and 3 parameters) の出口	152
0x9204	receive() の入口	0
0x9214	receive() の出口	152
0x9205	receive(long timeout) の入口	0
0x9215	receive(long timeout) の出口	152
0x9206	receiveNoWait() の入口	0
0x9216	receiveNoWait() の出口	152
0x9207	nextElement() の入口	0
0x9217	nextElement() の出口	152
0x92A0	キュー監視スレッド・MQCONN の入口	0
0x92B0	キュー監視スレッド・MQCONN の出口	152
0x92A1	キュー監視スレッド・MQOPEN の入口	0

イベント ID	トレース取得ポイント	詳細情報データ長（単位：バイト）
0x92B1	キュー監視スレッド・MQOPEN の出口	152
0x92A2	キュー監視スレッド・MQGET(検索) の入口※ 1	0
0x92B2	キュー監視スレッド・MQGET(検索) の出口※ 2	152
0x92A3	キュー監視スレッド・MQGET(削除) の入口	0
0x92B3	キュー監視スレッド・MQGET(削除) の出口	152
0x92A4	キュー監視スレッド・MQGET(ガーベッジ) の入口	0
0x92B4	キュー監視スレッド・MQGET(ガーベッジ) の出口	152
0x92A5	キュー監視スレッド・MQCLOSE の入口	0
0x92B5	キュー監視スレッド・MQCLOSE の出口	152
0x92A6	キュー監視スレッド・MQDISC の入口	0
0x92B6	キュー監視スレッド・MQDISC の出口	152
0x92A7	キュー監視スレッド・MQBEGIN の入口	0
0x92B7	キュー監視スレッド・MQBEGIN の出口	152
0x92A8	キュー監視スレッド・MQCMIT の入口	0
0x92B8	キュー監視スレッド・MQCMIT の出口	152
0x92A9	キュー監視スレッド・MQBACK の入口	0
0x92B9	キュー監視スレッド・MQBACK の出口	152
0x92AA	キュー監視スレッド・onMessage の入口	0
0x92BA	キュー監視スレッド・onMessage の出口	152
0x92AB	キュー監視スレッド・beforeDelivery の入口	0
0x92BB	キュー監視スレッド・beforeDelivery の出口	152
0x92AC	キュー監視スレッド・afterDelivery の入口	0
0x92BC	キュー監視スレッド・afterDelivery の出口	152

注※ 1

前回の MQGET(検索) の理由コードに，MQRC_NO_MSG_AVAILABLE が返された場合，MQGET(検索) が繰り返し実行されますが，その場合，0x92A2 のトレースファイルは取得されません。

注※ 2

MQGET(検索) の理由コードに MQRC_NO_MSG_AVAILABLE が返された場合，スレッド終了時を除いて 0x92B2 のトレースファイルは取得されません。

（3）イベント ID ごとの取得内容

JMSPRF トレースファイルのイベント ID ごとの取得内容を，次の表に示します。メッ

セージ識別子または相関識別子によって，MQC サーバでのメッセージを特定できます。

表 2-15　JMSPRF トレースファイルのイベント ID ごとの取得内容

イベント ID	リターンコード	インタフェース名※2	オペレーション名※2	詳細情報※1					
				キュー名称(48)	メッセージ識別子(32)	相関識別子(32)	ユーザデータ(32)	コネクションハンドル(4)	オブジェクトハンドル(4)
0x9200	○	○	○	—	—	—	—	—	—
0x9210	○	○	○	○	○	○	○	○	▲
0x9201	○	○	○	—	—	—	—	—	—
0x9211	○	○	○	○	○	○	○	○	▲
0x9202	○	○	○	—	—	—	—	—	—
0x9212	○	○	○	○	○	○	○	○	△
0x9203	○	○	○	—	—	—	—	—	—
0x9213	○	○	○	○	○	○	○	○	△
0x9204	○	○	○	—	—	—	—	—	—
0x9214	○	○	○	○	○	○	○	○	▲
0x9205	○	○	○	—	—	—	—	—	—
0x9215	○	○	○	○	○	○	○	○	▲
0x9206	○	○	○	—	—	—	—	—	—
0x9216	○	○	○	○	○	○	○	○	▲
0x9207	○	○	○	—	—	—	—	—	—
0x9217	○	○	○	○	○	○	○	○	▲
0x92A0	○	○	○	—	—	—	—	—	—
0x92B0	○	○	○	△	△	△	△	▲	△
0x92A1	○	○	○	—	—	—	—	—	—
0x92B1	○	○	○	○	△	△	△	○	▲
0x92A2	○	○	○	—	—	—	—	—	—
0x92B2	○	○	○	○	▲	▲	▲	○	○
0x92A3	○	○	○	—	—	—	—	—	—
0x92B3	○	○	○	○	▲	▲	▲	○	○
0x92A4	○	○	○	—	—	—	—	—	—

イベント ID	リターンコード	インタフェース名※2	オペレーション名※2	詳細情報※1					
				キュー名称(48)	メッセージ識別子(32)	相関識別子(32)	ユーザデータ(32)	コネクションハンドル(4)	オブジェクトハンドル(4)
0x92B4	○	○	○	○	△	△	△	○	○
0x92A5	○	○	○	―	―	―	―	―	―
0x92B5	○	○	○	○	△	△	△	○	○
0x92A6	○	○	○	―	―	―	―	―	―
0x92B6	○	○	○	△	△	△	△	○	△
0x92A7	○	○	○	―	―	―	―	―	―
0x92B7	○	○	○	△	△	△	△	○	△
0x92A8	○	○	○	―	―	―	―	―	―
0x92B8	○	○	○	△	△	△	△	○	△
0x92A9	○	○	○	―	―	―	―	―	―
0x92B9	○	○	○	△	△	△	△	○	△
0x92AA	○	○	○	―	―	―	―	―	―
0x92BA	○	○	○	△	△	△	△	○	△
0x92AB	○	○	○	―	―	―	―	―	―
0x92BB	○	○	○	△	△	△	△	○	△
0x92AC	○	○	○	―	―	―	―	―	―
0x92BC	○	○	○	△	△	△	△	○	△

（凡例）

○：情報を取得します。

▲：情報を取得しますが，エラーの場合は無効な値です。

△：情報を取得しますが，常に無効な値です。

―：情報を取得しません。

注※ 1

cprfed コマンドに -Dump または -CSV を指定した場合に出力される情報，および出力順序を示します。括弧内は，取得バイト数です。

注※ 2

出力される文字数の詳細については，マニュアル「Cosminexus リファレンス コマンド編」の「cprfed（性能解析トレース情報の編集出力）」の「出力形式」を参照してください。

(4) 表示形式

(a) JMSPRF トレース情報の表示形式

取得したトレース情報は cprfed コマンドで出力される次の形式に対応します。

- **リターンコード**
  リターンコードは,「Rc」に表示されます。

イベント ID	トレース取得ポイント	リターンコード
0x920	入口	0
0x921，0x922	正常出口	1
	異常出口	-1
0x92B0 ～ 0x92B9	正常出口	0
	異常出口	（各 MQI の理由コード）
0x92BA ～ 0x92BC	正常出口	0
	異常出口	-1

- **インタフェース名**
  インタフェース名は,「INT」に「MQAccess: メソッド名」の形式で表示されます。

イベント ID	インタフェース名
0x9200，0x9210	MQAccess:send(message only)
0x9201，0x9211	MQAccess:send(msg, 3param)
0x9202，0x9212	MQAccess:send(queue, message)
0x9203，0x9213	MQAccess:send(que, msg, 3param)
0x9204，0x9214	MQAccess:receive(no argument)
0x9205，0x9215	MQAccess:receive(timeout)
0x9206，0x9216	MQAccess:receiveNoWait
0x9207，0x9217	MQAccess:browse(nextElement)
0x92A0，0x92B0	MQAccess:run MQCONN
0x92A1，0x92B1	MQAccess:run MQOPEN
0x92A2，0x92B2	MQAccess:run MQGET(browse)
0x92A3，0x92B3	MQAccess:run MQGET(delete)
0x92A4，0x92B4	MQAccess:run MQGET(garbage)
0x92A5，0x92B5	MQAccess:run MQCLOSE
0x92A6，0x92B6	MQAccess:run MQDISC

イベント ID	インタフェース名
0x92A7，0x92B7	MQAccess:run MQBEGIN
0x92A8，0x92B8	MQAccess:run MQCMIT
0x92A9，0x92B9	MQAccess:run MQBACK
0x92AA，0x92BA	MQAccess:run onMessage
0x92AB，0x92BB	MQAccess:run beforeDelivery
0x92AC，0x92BC	MQAccess:run afterDelivery

- **オペレーション名**
  オペレーション名は，「OPR」に「オブジェクト名 & ハッシュコード」の形式で表示されます。

イベント ID	オペレーション名
0x9200，0x9210，0x9201，0x9211，0x9202，0x9212，0x9203，0x9213	jp.co.Hitachi.soft.mqadaptor.QueueSenderImpl@00000000 【出力文字列※】*adaptor.QueueSenderImpl@00000000
0x9204，0x9214，0x9205，0x9215，0x9206，0x9216	jp.co.Hitachi.soft.mqadaptor.QueueReceiverImpl@00000000 【出力文字列※】*aptor.QueueReceiverImpl@00000000
0x9207，0x9217	jp.co.Hitachi.soft.mqadaptor.MessageEnumeration@00000000 【出力文字列※】*ptor.MessageEnumeration@00000000
0x92A0，0x92B0，0x92A1，0x92B1，0x92A2，0x92B2，0x92A3，0x92B3，0x92A4，0x92B4，0x92A5，0x92B5，0x92A6，0x92B6，0x92A7，0x92B7，0x92A8，0x92B8，0x92A9，0x92B9，0x92AA，0x92BA，0x92AB，0x92BB，0x92AC，0x92BC	jp.co.Hitachi.soft.mqadaptor.ThreadController@00000000 【出力文字列※】*daptor.ThreadController@00000000

注※
オブジェクトの文字列表現の後ろ 32 文字が出力されます。

(b) JMSPRF トレースファイル個別情報の表示形式

cprfed コマンドの -Dump オプション（詳細情報をダンプ形式で出力），または -CSV オプション（詳細情報を CSV 形式で出力）で表示される詳細情報は，OPT に連続して表示されます。

- **キュー名称**
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa（キュー名称（48バイト））

合計48バイト
キュー名称が48バイト以下の場合，後ろが「¥0」で埋められます。

- **メッセージ識別子**
  bbbbbbbbbbbbbbbbbbbbbbbb（メッセージ識別子（24バイト） + ”¥0¥0¥0¥0¥0¥0¥0¥0 ”）
  合計32バイト
  接頭辞が「ID:」で始まる場合，取得する情報は「ID:」以降の24バイトとなります。
- **相関識別子**
  cccccccccccccccccccccccc（相関識別子（24バイト） + ”¥0¥0¥0¥0¥0¥0¥0¥0 ”）
  合計32バイト
  接頭辞が「ID:」で始まる場合，取得する情報は「ID:」以降の24バイトとなります。
- **ユーザデータ**
  ddddddddddddddddddddddddddddddddd（ユーザデータ先頭32バイト）
  合計32バイト
  ユーザデータが32バイト以下の場合，後ろが「¥0」で埋められます。
- **コネクションハンドル**
  eeeeeeee（コネクションハンドル，4バイト16進情報）
  合計4バイト
- **オブジェクトハンドル**
  ffffffff（オブジェクトハンドル，4バイト16進情報）
  合計4バイト

3 MQC クライアント機能の運用コマンド

この章では，MQC クライアント機能の運用コマンドについて説明します。

MQC クライアント機能の運用コマンドの概要

mqcapiout（API トレースファイル取得）

MQC クライアント機能の運用コマンドの概要

MQC クライアント機能の運用コマンド一覧

MQC クライアント機能の運用コマンドについて次の表に示します。コマンドの詳細については，以降で説明します。

表 3-1　MQC クライアント機能の運用コマンド

機能	コマンド名称	オフライン中に実行	オンライン中に実行	アクセス権
API トレースファイル取得	mqcapiout	○	×	TP1/Message Queue - Access 管理者

（凡例）
○：実行できます。
×：実行できません。

コマンド実行時の注意事項

Windows の TP1/Message Queue - Access を使用する場合，次に示す注意事項があります。

- 複数のパス名の区切り文字が，UNIX と Windows とで異なります。UNIX ではコロン（：）が複数のパス名の区切り文字ですが，Windows ではコロンはドライブ名とディレクトリ名との区切り文字になります。Windows ではパス名の区切り文字にセミコロン（；）を使用してください。
- パス名を完全パスで指定する際には，必ずドライブ名を記述してください。
- OpenTP1 は大文字と小文字を区別します。コマンドのオプションや，定義ファイルに記述した文字列をコマンド引数で使用するような場合は，注意してください。
  例えば，-a オプションと -A オプションでは異なるオプションを表します。
- ファイル名を指定するときは，ドライブ名以外にコロン（：）を含んだファイル名を指定しないでください。

mqcapiout（API トレースファイル取得）

形式

```
mqcapiout 〔 { -i スレッドID | -k コネクションハンドル | -x | -s | -t } 〕
          { トレースファイル名 | -c コアファイル名 }
```

機能

UAP が異常終了した場合に，API トレースファイルまたはコアファイルから API トレースファイル情報を取得します。〔 〕内のすべてのオプションを省略した場合は，すべての API トレース情報がスレッド ID 単位に出力されます。

オプション

● -i スレッド ID ～＜ 10 進数＞

指定されたスレッド ID の API トレース情報を出力します。スレッド ID は，-x オプションで出力される ID を指定します。

● -k コネクションハンドル ～ ((0x00000001 ～ 0x7FFFFFFF ※))

指定されたコネクションハンドルの API トレース情報を出力します。コネクションハンドルは，-s オプションで出力されるコネクションハンドルをコピー（複写）& ペースト（貼り付け）で指定します。ただし，0x00000001 ～ 0x7FFFFFFF ※以外の値を指定した場合は，メッセージ KFCA30971-E を出力します。

注※

64bit 版 OS では，0x0000000000000001 ～ 0x000000007FFFFFFF になります。

● -x

スレッド ID を一覧で出力します。

● -s

MQI 命令の成功および失敗※に関係なく，トレースファイル内にあるすべてのコネクションハンドルを一覧で出力します。MQCONN の入口情報と MQDISC の出口情報はコネクションハンドルを持たないため，API トレースファイルにこれら 2 種類のトレース情報しかない場合は，メッセージ KFCA30982-E を出力します。

注※

コネクションハンドルの値が不定（0x00000000 ～ 0xFFFFFFFF）になります。ただし，64bit 版 OS では，0x0000000000000000 ～ 0x00000000FFFFFFFF になります。

● -t

API トレース情報を取得日時の順番で出力します。

● -c コアファイル名

UAP が異常終了した場合に出力されるコアファイル名を指定します。

コマンド引数

● トレースファイル名 ～＜文字列＞

UAP が異常終了した場合に出力される API トレースファイル名を指定します。

API トレースファイルは，API トレースディスク出力要否が ON の場合，UAP を実行したディレクトリに次の形式で出力します。

```
mqc.api.ppp.n
```

ppp：プロセス ID
n：0 ～指定ファイル数 -1

出力形式

次の文字コードをシステム環境 LANG に設定している場合，API トレースの編集結果は日本語で出力します。そのほかの認識できない環境の場合は，英語で出力します。

- HP-UX の場合：ja_JP.SJIS
- AIX の場合：Ja_JP
- Windows の場合：ja_JP.SJIS
- Red Hat Enterprise Linux AS 3（x86/IPF）の場合：ja_JP.eucJP または ja_JP
- Red Hat Enterprise Linux AS 4/ES 4（x86）の場合：ja_JP.UTF-8，ja_JP.UTF8，ja_JP.utf-8，または ja_JP.utf8

出力形式（MQCONN の場合）

```
関数　＝　MQCONN（出口）
            取得日時　＝　yy/mm/dd hh:mm:ss
            マイクロ秒　＝　AA....AA
            トレース取得通番　＝　BB....BB
            スレッドID　＝　CC....CC
            キューマネジャー名　＝　DD....DD
            コネクションハンドル　＝　[0x EE....EE]※1
            完了コード　＝　FF....FF（GG....GG）※1
            理由コード　＝　HH....HH（GG....GG）※1
```

注※ 1
出口情報を表示するときだけ出力します。

● 出力項目の説明

yy/mm/dd
　取得した年（西暦下 2 けた）月日（半角数字 8 文字）です。

hh:mm:ss
　取得した時分秒（半角数字 8 文字）です。

AA....AA
　取得日時のマイクロ秒（10 進数）です。

BB....BB
　トレース取得通番（10 進数）です。

CC....CC
　スレッド ID（10 進数）です。

DD....DD
　キューマネジャ名（文字列）です。

EE....EE
　コネクションハンドル（16 進数）です。

FF....FF
　完了コード（10 進数）です。

GG....GG
　定数名です。

HH....HH
　理由コード（10 進数）です。

出力形式（MQOPEN の場合）

```
関数　＝　MQOPEN（出口）
        取得日時　＝　yy/mm/dd hh:mm:ss
        マイクロ秒　＝　AA....AA
        トレース取得通番　＝　BB....BB
        スレッドID　＝　CC....CC
        コネクションハンドル　＝　[0xDD....DD]
        オブジェクトハンドル　＝　[0xEE....EE]※1
        構造体識別子　＝　FF....FF（GG....GG）
        構造体バージョン番号　＝　HH....HH（GG....GG）
        オブジェクトタイプ　＝　II....II（GG....GG）
        オブジェクト名　＝　JJ....JJ
        オブジェクトキューマネジャー名　＝　KK....KK
        動的キュー名　＝　LL....LL
        代替ユーザ識別子　＝　MM....MM
        動作オプション　＝　NN....NN（GG....GG）
                                    :
```

```
                                        (GG....GG)
  完了コード　＝　OO....OO (MQCC_OK)※1
  理由コード　＝　PP....PP (MQRC_NON)※1
  オブジェクトレコード数　＝　QQ....QQ※2※3
  ローカルキューの数　＝　RR....RR※1※2※3
  リモートキューの数　＝　SS....SS※1※2※3
  オープンに失敗したキューの数　＝　TT....TT※1※2※3
  最初のオブジェクトレコードまでのオフセット　＝　UU....UU※2※3
  最初の応答レコードまでのオフセット　＝　VV....VV※2※3
  最初のオブジェクトレコードのアドレス　＝　[0x WW....WW]※2※3
  最初の応答レコードのアドレス　＝　[0x XX....XX]※2※3
   ------ 代替セキュリティ識別子 ------※3
 000000  [0x YY....YY YY....YY YY....YY YY....YY]  aa....aa※3
 000010  [0x YY....YY YY....YY YY....YY YY....YY]  aa....aa※3
  000020  [0x YY....YY YY....YY]  aa....aa※3
  解決したキュー名称　＝　bb....bb※3
  解決したキューマネジャー名称　＝　cc....cc※3
```

注※ 1

出口情報を表示するときだけ出力します。

注※ 2

MQOD 構造体バージョン番号が MQOD_VERSION_2 の場合に出力します。

注※ 3

MQOD 構造体バージョン番号が MQOD_VERSION_3 の場合に出力します。

● 出力項目の説明

`yy/mm/dd`

取得した年（西暦下 2 けた）月日（半角数字 8 文字）です。

`hh:mm:ss`

取得した時分秒（半角数字 8 文字）です。

`AA....AA`

取得日時のマイクロ秒（10 進数）です。

`BB....BB`

トレース取得通番（10 進数）です。

`CC....CC`

スレッド ID（10 進数）です。

`DD....DD`

コネクションハンドル（16 進数）です。

EE....EE
　オブジェクトハンドル（16 進数）です。

FF....FF
　構造体識別子（文字列）です。

GG....GG
　定数名です。

HH....HH
　構造体バージョン番号（10 進数）です。

II....II
　オブジェクトタイプ（10 進数）です。

JJ....JJ
　オブジェクト名（文字列）です。

KK....KK
　オブジェクトキューマネジャ名（文字列）です。

LL....LL
　動的キュー名（文字列）です。

MM....MM
　代替ユーザ識別子（文字列）です。

NN....NN
　動作オプション（10 進数）です。

OO....OO
　完了コード（10 進数）です。

PP....PP
　理由コード（10 進数）です。

QQ....QQ
　オブジェクトレコード数（10 進数）です。

RR....RR
　ローカルキュー数（10 進数）です。

SS....SS
　リモートキュー数（10 進数）です。

TT....TT
　オープンに失敗したキューの数（10 進数）です。

UU....UU
　オブジェクトレコードまでのオフセット（16 進数）です。

VV....VV
　応答レコードまでのオフセット（16 進数）です。

WW....WW
　オブジェクトレコードのアドレス（16 進数）です。

XX....XX
　応答レコードのアドレス（16 進数）です。

YY....YY
　代替セキュリティ識別子の内容（16 進数）です。

aa....aa
　代替セキュリティ識別子の内容（文字列）です。

bb....bb
　キュー名称（文字列）です。

cc....cc
　キューマネジャ名称（文字列）です。

出力形式（MQPUT の場合）

```
関数　＝　MQPUT（出口）
          取得日時　＝　yy/mm/dd hh:mm:ss
          マイクロ秒　＝　AA....AA
          トレース取得通番　＝　BB....BB
          スレッドID　＝　CC....CC
          コネクションハンドル　＝　[0x DD....DD]
          オブジェクトハンドル　＝　[0x EE....EE]
          バッファ長　＝　FF....FF
          ------　バッファの内容　------
          000000    [0x GG....GG GG....GG GG....GG]    HH....HH
          完了コード　＝　II....II（JJ....JJ）※1
          理由コード　＝　KK....KK(JJ....JJ)※1
          構造体識別子　＝　LL....LL　（JJ....JJ）
          構造体バージョン番号　＝　MM....MM（JJ....JJ）
          報告オプション　＝　NN....NN(JJ....JJ)
          メッセージタイプ　＝　OO....OO(JJ....JJ)
          メッセージ保持時間　＝　PP....PP（JJ....JJ）
          返答コード　＝　QQ....QQ（JJ....JJ）
          マシンコード形式　＝　RR....RR(JJ....JJ)
          文字セット識別子　＝　SS....SS
          フォーマット名　＝　TT....TT　　(JJ....JJ)
          メッセージ優先度　＝　UU....UU（JJ....JJ）
          メッセージ永続性　＝　VV....VV（JJ....JJ）
          ------　メッセージ識別子　------
        000000   [0x WW....WW WW....WW WW....WW WW....WW]   XX....XX
```

```
  000010    [0x WW....WW WW....WW]    XX....XX
  ------  相関識別子  ------
000000   [0x YY....YY YY....YY YY....YY YY....YY]   aa....aa
  000010    [0x YY....YY YY....YY ]    aa....aa
  ロールバックカウンタ  ＝  bb... bb
  応答先キュー名  ＝  cc....cc
  応答先キューマネジャー名  ＝  dd....dd
  ユーザ識別子  ＝  ee....ee
  ------  課金トークン  ------
000000   [0x ff....ff ff....ff ff....ff ff....ff]   gg....gg
000010   [0x ff....ff ff....ff ff....ff ff....ff]   gg....gg
  識別データ  ＝  hh....hh
  登録アプリケーションタイプ  ＝  ii....ii (JJ....JJ)
  登録アプリケーション名  ＝  jj....jj
  登録日時  ＝  kk....kk ll....ll
  登録元データ  ＝  mm....mm

  ------  グループ識別子  ------※2
000000   [0x nn....nn nn....nn nn....nn nn....nn]   oo....oo
※2
  000010    [0x nn....nn nn....nn]    oo....oo※2
  メッセージシーケンス番号  ＝  pp....pp※2
  オフセット  ＝  qq....qq※2
  メッセージフラグ  ＝  rr....rr (JJ....JJ)※2
  登録元メッセージ長  ＝  ss....ss※2

  MQPMO構造体識別子  ＝  MM....MM  (JJ....JJ)
  MQPMO構造体バージョン番号  ＝  MM....MM (JJ....JJ)
  オプション  ＝  tt....tt (JJ....JJ)
  コンテキスト  ＝  [0x uu....uu]
  ローカルキューの数  ＝  vv....vv
  リモートキューの数  ＝  ww....ww
  送信に失敗したキューの数  ＝  xx....xx

  受信キュー名  ＝  yy....yy
  受信キューマネジャー名  ＝  zz....zz
 登録メッセージレコード数または応答メッセージレコード数  ＝  a1....a1
※3
  MQPMR構造体フィールドの存在  ＝  b1....b1 (JJ....JJ)※3
  最初の登録メッセージレコードのオフセット  ＝  c1....c1※3
  最初の応答レコードのオフセット  ＝  d1....d1※3
  最初の登録メッセージレコードのアドレス  ＝  [0x e1....e1]※3
  最初の応答レコードのアドレス  ＝  [0x f1....f1]※3
```

注※ 1

出口情報を表示するときだけ出力します。

注※ 2

MQMD 構造体バージョン番号が MQMD_VERSION_2 の場合に出力します。

注※ 3

MQPMO 構造体バージョン番号が MQPMO_VERSION_2 の場合に出力します。

● 出力項目の説明

yy/mm/dd
　取得した年（西暦下 2 けた）月日（半角数字 8 文字）です。

hh:mm:ss
　取得した時分秒（半角数字 8 文字）です。

AA....AA
　取得日時のマイクロ秒（10 進数）です。

BB....BB
　トレース取得通番（10 進数）です。

CC....CC
　スレッド ID（10 進数）です。

DD....DD
　コネクションハンドル（16 進数）です。

EE....EE
　オブジェクトハンドル（16 進数）です。

FF....FF
　バッファ長（10 進数）です。単位はバイトです。

GG....GG
　バッファの内容（16 進数）です。
　バッファ長が 0 バイトのときは出力しません。バッファ長が 32 バイトを超えるときは先頭から 32 バイトまでを出力します。

HH....HH
　バッファの内容（16 進数）です。
　バッファ長が 0 バイトのときは出力しません。バッファ長が 32 バイトを超えるときは先頭から 32 バイトまでを出力します。

II....II
　完了コード（10 進数）です。

JJ....JJ
　定数名です。

KK....KK
　理由コード（10 進数）です。

LL....LL
構造体識別子（文字列）です。

MM....MM
構造体バージョン番号（10 進数）です。

NN....NN
報告オプション（10 進数）です。

OO....OO
メッセージタイプ（10 進数）です。

PP....PP
メッセージ保持時間（10 進数）です。

QQ....QQ
返答コード（10 進数）です。

RR....RR
マシンコード形式（10 進数）です。

SS....SS
文字セット識別子（10 進数）です。

TT....TT
フォーマット名（文字列）です。

UU....UU
メッセージ優先度（10 進数）です。

VV....VV
メッセージ永続性（10 進数）です。

WW....WW
メッセージ識別子の内容（16 進数）です。

XX....XX
メッセージ識別子の内容（文字列）です。

YY....YY
相関識別子の内容（16 進数）です。

aa....aa
相関識別子の内容（文字列）です。

bb....bb
ロールバックカウンタ（10 進数）です。

cc....cc
応答先キュー名（文字列）です。

dd....dd
応答先キューマネジャ名（文字列）です。

ee....ee
ユーザ識別子（文字列）です。

ff....ff
課金トークンの内容（16 進数）です。

gg....gg
課金トークンの内容（文字列）です。

hh....hh
識別データ（文字列）です。

ii....ii
登録アプリケーションタイプ（10 進数）です。

jj....jj
登録アプリケーション名（文字列）です。

kk....kk
登録日（半角数字 8 文字）です。

ll....ll
登録時刻（半角数字 8 文字）です。

mm....mm
登録元データ（文字列）です。

nn....nn
グループ識別子の内容（16 進数）です。

oo....oo
グループ識別子の内容（文字列）です。

pp....pp
メッセージシーケンス番号（10 進数）です。

qq....qq
オフセット（16 進数）です。

rr....rr
メッセージフラグ（10 進数）です。

ss....ss
　登録元メッセージ長（10 進数）です。単位はバイトです。

tt....tt
　オプション（10 進数）です。

uu....uu
　コンテキスト（16 進数）です。

vv....vv
　ローカルキューの数（10 進数）です。

ww....ww
　リモートキューの数（10 進数）です。

xx....xx
　送信に失敗したキューの数（10 進数）です。

yy....yy
　受信キュー名（文字列）です。

zz....zz
　受信キューマネジャ名（文字列）です。

a1....a1
　レコード数（10 進数）です。

b1....b1
　MQPMR 構造体フィールドの存在（10 進数）です。

c1....c1
　登録メッセージレコードのオフセット（16 進数）です。

d1....d1
　応答レコードのオフセット（16 進数）です。

e1....e1
　登録メッセージレコードのアドレス（16 進数）です。

f1....f1
　応答レコードアドレス（16 進数）です。

出力形式（MQGET の場合）

```
関数　＝　MQGET（出口）
　　取得日時　＝　yy/mm/dd hh:mm:ss
　　マイクロ秒　＝　AA....AA
```

```
  トレース取得通番 ＝ BB....BB
  スレッドID ＝ CC....CC
  コネクションハンドル ＝ [0x DD....DD]
  オブジェクトハンドル ＝ [0x EE....EE]
  バッファ長 ＝ FF....FF

  ------ バッファの内容 ------※1
000000  [0x GG....GG GG....GG GG....GG GG....GG]  HH....HH
※1
000010  [0x GG....GG GG....GG GG....GG GG....GG]  HH....HH
※1
  完了コード ＝ II....II (JJ....JJ)※1
  理由コード ＝ KK....KK (JJ....JJ)※1
  構造体識別子 ＝ LL....LL  (JJ....JJ)
  構造体バージョン番号 ＝ MM....MM (JJ....JJ)
  報告オプション ＝ NN....NN (JJ....JJ)
  メッセージタイプ ＝ OO....OO (JJ....JJ)
  メッセージ保持時間 ＝ PP....PP (JJ....JJ)
  返答コード ＝ QQ....QQ (JJ....JJ)
  マシンコード形式 ＝ RR....RR (JJ....JJ)
  文字セット識別子 ＝ SS....SS
  フォーマット名 ＝ TT....TT  (JJ....JJ)
  メッセージ優先度 ＝ UU....UU
  メッセージ永続性 ＝ VV....VV (JJ....JJ)

  ------ メッセージ識別子 ------
000000  [0x WW....WW WW....WW WW....WW WW....WW]  XX....XX
  000010  [0x WW....WW WW....WW]  XX....XX

  ------ 相関識別子 ------
000000  [0x YY....YY YY....YY YY....YY YY....YY]  aa....aa
  000010  [0x YY....YY YY....YY]  aa....aa
  ロールバックカウンタ ＝ bb....bb
  応答先キュー名 ＝ cc....cc
  応答先キューマネジャー名 ＝ dd....dd
  ユーザ識別子 ＝ ee....ee

  ------ 課金トークン ------
000000  [0x ff....ff ff....ff ff....ff ff....ff]  gg....gg
000010  [0x ff....ff ff....ff ff....ff ff....ff]  gg....gg
  識別データ ＝ hh....hh
  登録アプリケーションタイプ ＝ ii....ii (JJ....JJ)
  登録アプリケーション名 ＝ jj....jj
  登録日時 ＝ kk....kk ll....ll
  登録元データ ＝ mm....mm

  ------ グループ識別子 ------
000000  [0x nn....nn nn....nn nn....nn nn....nn] oo....oo※2
  000010  [0x nn....nn nn....nn]  oo....oo※2
  メッセージシーケンス番号 ＝ pp....pp※2
  オフセット ＝ qq....qq※2
  メッセージフラグ ＝ rr....rr (JJ....JJ)※2
  登録元メッセージ長 ＝ ss....ss※2
  MQGMO構造体識別子 ＝ LL....LL(JJ....JJ)
  MQGMO構造体バージョン番号 ＝ MM....MM (JJ....JJ)
  オプション ＝ tt....tt  (JJ....JJ)
                          :
```

```
                                    (JJ....JJ)
待ち合わせ最大時間　＝　uu....uu
シグナル　＝　vv....vv
受信キュー名　＝　ww....ww
メッセージ長　＝　xx....xx

一致オプション　＝　yy....yy（JJ....JJ）※3
メッセージグループフラグ　＝　zz....zz（JJ....JJ）※3
論理メッセージフラグ　＝　a1....a1（JJ....JJ）※3
セグメント分割フラグ　＝　b1....b1（JJ....JJ）※3
```

注※ 1

出口情報を表示するときだけ出力します。

注※ 2

MQMD 構造体バージョン番号が MQMD_VERSION_2 の場合に出力します。

注※ 3

MQGMO 構造体バージョン番号が MQGMO_VERSION_2 の場合に出力します。

● 出力項目の説明

yy/mm/dd

取得した年（西暦下 2 けた）月日（半角数字 8 文字）です。

hh:mm:ss

取得した時分秒（半角数字 8 文字）です。

AA....AA

取得日時のマイクロ秒（10 進数）です。

BB....BB

トレース取得通番（10 進数）です。

CC....CC

スレッド ID（10 進数）です。

DD....DD

コネクションハンドル（16 進数）です。

EE....EE

オブジェクトハンドル（16 進数）です。

FF....FF

バッファ長（10 進数）です。単位はバイトです。

GG....GG

バッファの内容（16 進数）です。

バッファ長が 0 バイトのときは出力しません。バッファ長が 32 バイトを超えるときは先頭から 32 バイトまでを出力します。

HH....HH
バッファの内容（文字列）です。
バッファ長が 0 バイトのときは出力しません。バッファ長が 32 バイトを超えるときは先頭から 32 バイトまでを出力します。

II....II
完了コード（10 進数）です。

JJ....JJ
定数名です。

KK....KK
理由コード（10 進数）です。

LL....LL
構造体識別子（文字列）です。

MM....MM
構造体バージョン番号（10 進数）です。

NN....NN
報告オプション（10 進数）です。

OO....OO
メッセージタイプ（10 進数）です。

PP....PP
メッセージ保持時間（10 進数）です。

QQ....QQ
返答コード（10 進数）です。

RR....RR
マシンコード形式（10 進数）です。

SS....SS
文字セット識別子（10 進数）です。

TT....TT
フォーマット名（文字列）です。

UU....UU
メッセージ優先度（10 進数）です。

VV....VV
メッセージ永続性（10 進数）です。

WW....WW
メッセージ識別子の内容（16 進数）です。

XX....XX
メッセージ識別子の内容（文字列）です。

YY....YY
相関識別子の内容（16 進数）です。

aa....aa
相関識別子の内容（文字列）です。

bb....bb
ロールバックカウンタ（10 進数）です。

cc....cc
応答先キュー名（文字列）です。

dd....dd
応答先キューマネジャ名（文字列）です。

ee....ee
ユーザ識別子（文字列）です。

ff....ff
課金トークンの内容（16 進数）です。

gg....gg
課金トークンの内容（文字列）です。

hh....hh
識別データ（文字列）です。

ii....ii
登録アプリケーションタイプ（10 進数）です。

jj....jj
登録アプリケーション名（文字列）です。

kk....kk
登録日（半角数字 8 文字）です。

ll....ll
登録時刻（半角数字 8 文字）です。

mm....mm
登録元データ（文字列）です。

nn....nn
グループ識別子の内容（16 進数）です。

oo....oo
グループ識別子の内容（文字列）です。

pp....pp
メッセージシーケンス番号（10 進数）です。

qq....qq
オフセット（16 進数）です。

rr....rr
メッセージフラグ（10 進数）です。

ss....ss
登録元メッセージ長（10 進数）です。単位はバイトです。

tt....tt
オプション（10 進数）です。

uu....uu
待ち合わせ最大時間（10 進数）です。

vv....vv
シグナル（10 進数）です。

ww....ww
受信キュー名（文字列）です。

xx....xx
メッセージ長（10 進数）です。単位はバイトです。

yy....yy
一致オプション（10 進数）です。

zz....zz
メッセージグループフラグ（文字列）です。

a1....a1
論理メッセージフラグ（文字列）です。

b1....b1
セグメント分割フラグ（文字列）です。

出力形式（MQCLOSE の場合）

```
関数 ＝ MQCLOSE（出口）
          取得日時 ＝ yy/mm/dd hh:mm:ss
          マイクロ秒 ＝ AA....AA
          トレース取得通番 ＝ BB....BB
          スレッドID ＝ CC....CC
          コネクションハンドル ＝ [0x DD....DD]
          オブジェクトハンドル ＝ [0x EE....EE]
          動作オプション ＝ FF....FF（GG....GG）
          完了コード ＝ HH....HH(GG....GG)※1
          理由コード ＝ II....II（GG....GG）※1
```

注※ 1

出口情報を表示するときだけ出力します。

● 出力項目の説明

yy/mm/dd

取得した年（西暦下 2 けた）月日（半角数字 8 文字）です。

hh:mm:ss

取得した時分秒（半角数字 8 文字）です。

AA....AA

取得日時のマイクロ秒（10 進数）です。

BB....BB

トレース取得通番（10 進数）です。

CC....CC

スレッド ID（10 進数）です。

DD....DD

コネクションハンドル（16 進数）です。

EE....EE

オブジェクトハンドル（16 進数）です。

FF....FF

動作オプション（10 進数）です。

GG....GG

定数名です。

HH....HH

完了コード（10 進数）です。

II....II
　理由コード（10 進数）です。

出力形式（MQDISC の場合）

```
関数　＝　MQDISC（出口）
　　取得日時　＝　yy/mm/dd hh:mm:ss
　　マイクロ秒　＝　AA....AA
　　トレース取得通番　＝　BB....BB
　　スレッドID　＝　CC....CC
　　コネクションハンドル　＝　[0x DD....DD]
　　完了コード　＝　EE....EE（FF....FF）※1
　　理由コード　＝　GG....GG（FF....FF）※1
```

注※ 1
　出口情報を表示するときだけ出力します。

● 出力項目の説明

yy/mm/dd
　取得した年（西暦下 2 けた）月日（半角数字 8 文字）です。

hh:mm:ss
　取得した時分秒（半角数字 8 文字）です。

AA....AA
　取得日時のマイクロ秒（10 進数）です。

BB....BB
　トレース取得通番（10 進数）です。

CC....CC
　スレッド ID（10 進数）です。

DD....DD
　コネクションハンドル（16 進数）です。

EE....EE
　完了コード（10 進数）です。

FF....FF
　定数名です。

GG....GG
　理由コード（10 進数）です。

出力形式（MQPUT1 の場合）

```
関数 ＝ MQPUT1（出口）
          取得日時 ＝ yy/mm/dd hh:mm:ss
           マイクロ秒 ＝ AA....AA
           トレース取得通番 ＝ BB....BB
           スレッドID ＝ CC....CC
           コネクションハンドル ＝ [0x DD....DD]
           バッファ長 ＝ EE....EE

           ------ バッファの内容 ------
           000000   [0x FF....FF FF....FF FF....FF FF....FF]
GG....GG
           000010   [0x FF....FF FF....FF FF....FF FF....FF]
GG....GG
           完了コード ＝ HH....HH(II....II)※1
           理由コード ＝ JJ....JJ (II....II)※1
           構造体識別子 ＝ KK....KK (II....II)
           構造体バージョン番号 ＝ LL....LL (II....II)
           報告オプション ＝ MM....MM (II....II)
           メッセージタイプ ＝ NN....NN (II....II)
           メッセージ保持時間 ＝ OO....OO (II....II)
           返答コード ＝ PP....PP (II....II)
           マシンコード形式 ＝ QQ....QQ (II....II)
           文字セット識別子 ＝ RR....RR (II....II)
           フォーマット名 ＝ SS....SS (II....II)
           メッセージ優先度 ＝ TT....TT (II....II)
           メッセージ永続性 ＝ UU....UU (II....II)

           ------ メッセージ識別子 ------
           000000   [0x VV....VV VV....VV VV....VV VV....VV]
WW....WW
           000010   [0x VV....VV VV....VV]   WW....WW

           ------ 相関識別子 ------
           000000   [0x XX....XX XX....XX XX....XX XX....XX]
YY....YY
           000010   [0x XX....XX XX....XX]   YY....YY
           ロールバックカウンタ ＝ aa....aa
           応答先キュー名 ＝ bb....bb
           応答先キューマネジャー名 ＝ cc....cc
           ユーザ識別子 ＝ dd....dd

           ------ 課金トークン ------
           000000   [0x ee....ee ee....ee ee....ee ee....ee]
ff....ff
           000010   [0x ee....ee ee....ee ee....ee ee....ee]
ff....ff
           識別データ ＝ gg....gg
           登録アプリケーションタイプ ＝ hh....hh (II....II)
           登録アプリケーション名 ＝ ii....ii
           登録日時 ＝ jj....jj kk....kk
           登録元データ ＝ ll....ll

           ------ グループ識別子 ------
           000000   [0x mm....mm mm....mm mm....mm mm....mm]
nn....nn
           000010   [0x mm....mm mm....mm]   nn....nn
           メッセージシーケンス番号 ＝ oo....oo
```

```
                オフセット　＝　pp....pp
                メッセージフラグ　＝　qq....qq (II....II)
                登録元メッセージ長　＝　rr....rr
                MQPMO構造体識別子　＝　KK....KK　(II....II)
                MQPMO構造体バージョン番号　＝　LL....LL (II....II)
                オプション　＝　ss....ss (II....II)
                コンテキスト　＝　[0x tt....tt]
                ローカルキューの数　＝　uu....uu
                リモートキューの数　＝　vv....vv
                送信に失敗したキューの数　＝　ww....ww
                受信キュー名　＝　xx....xx
                受信キューマネジャー名　＝　yy....yy
              登録メッセージレコード数または応答メッセージレコード数　＝　zz....zz
※2
                MQPMR構造体フィールドの存在　＝　a1....a1 (II....II)※2
                最初の登録メッセージレコードのオフセット　＝　b1....b1※2
                最初の応答レコードのオフセット　＝　c1....c1※2
                最初の登録メッセージレコードのアドレス　＝　[0x d1....d1]※2
                最初の応答レコードのアドレス　＝　[0x e1....e1]※2
                MQOD構造体識別子　＝　KK....KK　(II....II)
                MQOD構造体バージョン番号　＝　LL....LL (II....II)
                オブジェクトタイプ　＝　f1....f1
                オブジェクト名　＝　g1....g1
                オブジェクトキューマネジャー名　＝　h1....h1
                動的キュー名　＝　i1....i1
                代替ユーザ識別子　＝　j1....j1

                オブジェクトレコード数　＝　k1....k1※3※4
                ローカルキューの数　＝　uu....uu※1※3※4
                リモートキューの数　＝　vv....vv※1※3※4
                オープンに失敗したキューの数　＝　l1....l1※1※3※4
                最初のオブジェクトレコードまでのオフセット　＝　b1....b1※3※4
                最初の応答レコードまでのオフセット　＝　c1....c1※3※4
                最初のオブジェクトレコードのアドレス　＝　[0x d1....d1]※3※4
                最初の応答レコードのアドレス　＝　[0x e1....e1]※3※4

                ------ 代替セキュリティ識別子 ------※4
                000000    [0x m1....m1 m1....m1 m1....m1 m1....m1]
n1....n1※4
                000010    [0x m1....m1 m1....m1 m1....m1 m1....m1]
n1....n1※4
                000020    [0x m1....m1 m1....m1]   n1....n1※4
                解決したキュー名称　＝　o1....o1※4
                解決したキューマネジャー名称　＝　p1....p1※4
```

注※ 1

出口情報を表示するときだけ出力します。

注※ 2

MQPMO 構造体バージョン番号が MQPMO_VERSION_2 の場合に出力します。

注※ 3

MQOD 構造体バージョン番号が MQOD_VERSION_2 の場合に出力します。

注※ 4

MQOD 構造体バージョン番号が MQOD_VERSION_3 の場合に出力します。

● 出力項目の説明

yy/mm/dd

取得した年（西暦下 2 けた）月日（半角数字 8 文字）です。

hh:mm:ss

取得した時分秒（半角数字 8 文字）です。

AA....AA

取得日時のマイクロ秒（10 進数）です。

BB....BB

トレース取得通番（10 進数）です。

CC....CC

スレッド ID（10 進数）です。

DD....DD

コネクションハンドル（16 進数）です。

EE....EE

バッファ長（10 進数）です。単位はバイトです。

FF....FF

バッファの内容（16 進数）です。

バッファ長が 0 バイトのときは出力しません。バッファ長が 32 バイトを超えるときは先頭から 32 バイトまでを出力します。

GG....GG

バッファの内容（文字列）です。

バッファ長が 0 バイトのときは出力しません。バッファ長が 32 バイトを超えるときは先頭から 32 バイトまでを出力します。

HH....HH

完了コード（10 進数）です。

II....II

定数名です。

JJ....JJ

理由コード（10 進数）です。

KK....KK
　構造体識別子（文字列）です。

LL....LL
　構造体バージョン番号（10 進数）です。

MM....MM
　報告オプション（10 進数）です。

NN....NN
　メッセージタイプ（10 進数）です。

OO....OO
　メッセージ保持時間（10 進数）です。

PP....PP
　返答コード（10 進数）です。

QQ....QQ
　マシンコード形式（10 進数）です。

RR....RR
　文字セット識別子（10 進数）です。

SS....SS
　フォーマット名（文字列）です。

TT....TT
　メッセージ優先度（10 進数）です。

UU....UU
　メッセージ永続性（10 進数）です。

VV....VV
　メッセージ識別子の内容（16 進数）です。

WW....WW
　メッセージ識別子の内容（文字列）です。

XX....XX
　相関識別子の内容（16 進数）です。

YY....YY
　相関識別子の内容（文字列）です。

aa....aa
　ロールバックカウンタ（10 進数）です。

bb....bb
応答先キュー名（文字列）です。

cc....cc
応答先キューマネジャ名（文字列）です。

dd....dd
ユーザ識別子（文字列）です。

ee....ee
課金トークンの内容（16 進数）です。

ff....ff
課金トークンの内容（文字列）です。

gg....gg
識別データ（文字列）です。

hh..hh
登録アプリケーションタイプ（10 進数）です。

ii....ii
登録アプリケーション名（文字列）です。

jj....jj
登録日付（半角数字 8 文字）です。

kk....kk
登録時刻（半角数字 8 文字）です。

ll....ll
登録元データ（文字列）です。

mm....mm
グループ識別子の内容（16 進数）です。

nn....nn
グループ識別子の内容（文字列）です。

oo....oo
メッセージシーケンス番号（10 進数）です。

pp....pp
オフセット（10 進数）です。

qq....qq
メッセージフラグ（16 進数）です。

rr....rr
登録元メッセージ長（10 進数）です。単位はバイトです。

ss....ss
オプション（10 進数）です。

tt....tt
コンテキスト（16 進数）です。

uu....uu
ローカルキューの数（10 進数）です。

vv....vv
リモートキューの数（10 進数）です。

ww....ww
送信に失敗したキューの数（10 進数）です。

xx....xx
受信キュー名（文字列）です。

yy....yy
受信キューマネジャー名（文字列）です。

zz....zz
メッセージレコード数（10 進数）です。

a1....a1
MQPMR 構造体フィールドの存在です。

b1....b1
登録メッセージレコードのオフセットです。

c1....c1
応答レコードのオフセットです。

d1....d1
登録メッセージレコードのアドレス（16 進数）です。

e1....e1
応答レコードのアドレス（16 進数）です。

f1....f1
オブジェクトタイプ（10 進数）です。

g1....g1
オブジェクト名（文字列）です。

h1....h1
　オブジェクトキューマネジャ名（文字列）です。

i1....i1
　動的キュー名（文字列）です。

j1....j1
　代替ユーザ識別子（文字列）です。

k1....k1
　オブジェクトレコード数（10 進数）です。

l1....l1
　オープンに失敗したキューの数（10 進数）です。

m1....m1
　代替識別子の内容（16 進数）です。

n1....n1
　代替識別子の内容（文字列）です。

o1....o1
　キュー名称（文字列）です。

p1....p1
　キューマネジャ名称（文字列）です。

出力形式（MQINQ の場合）

```
関数　＝　MQINQ（出口）
        取得日時　＝　yy/mm/dd hh:mm:ss
        マイクロ秒　＝ AA....AA
        トレース取得通番　＝　BB....BB
        スレッドID　＝　CC....CC
        コネクションハンドル　＝　[0x DD....DD]
        オブジェクトハンドル　＝　[0x EE....EE]
        セレクタ数　＝　FF....FF
        セレクタ　＝　GG....GG（HH....HH）
                      GG....GG（HH....HH）
                                 :
                      GG....GG（HH....HH）
        整数型属性数　＝　II....II
        文字型属性長　＝　JJ....JJ
        完了コード　＝　KK....KK（HH....HH）※1
        理由コード　＝　LL....LL（HH....HH）※1
```

注※ 1
　出口情報を表示するときだけ出力します。

● 出力項目の説明

yy/mm/dd
　取得した年（西暦下 2 けた）月日（半角数字 8 文字）です。

hh:mm:ss
　取得した時分秒（半角数字 8 文字）です。

AA....AA
　取得日時のマイクロ秒（10 進数）です。

BB....BB
　トレース取得通番（10 進数）です。

CC....CC
　スレッド ID（10 進数）です。

DD....DD
　コネクションハンドル（16 進数）です。

EE....EE
　オブジェクトハンドル（16 進数）です。

FF....FF
　セレクタ数（10 進数）です。

GG....GG
　セレクタ（10 進数）です。

HH....HH
　定数名です。

II....II
　整数型属性数（10 進数）です。

JJ....JJ
　文字型属性長（10 進数）です。

KK....KK
　完了コード（10 進数）です。

LL....LL
　理由コード（10 進数）です。

出力形式（MQSET の場合）

関数　＝　MQSET（出口）

```
取得日時　＝　yy/mm/dd hh:mm:ss
マイクロ秒　＝　AA....AA
トレース取得通番　＝　BB....BB
スレッドID　＝　CC....CC
コネクションハンドル　＝　[0x DD....DD]
オブジェクトハンドル　＝　[0x EE....EE]
セレクタ数　＝　FF....FF
セレクタ　＝　GG....GG (HH....HH)
              GG....GG (HH....HH)
                          :
              GG....GG (HH....HH)
整数型属性数　＝　II....II
文字型属性長　＝　JJ....JJ
完了コード　＝　KK....KK (HH....HH)※1
理由コード　＝　LL....LL (HH....HH)※1
整数型属性値　＝　MM....MM
                  MM....MM
                     :
                  MM....MM
文字型属性値　＝　NN....NN
```

注※ 1

出口情報を表示するときだけ出力します。

● 出力項目の説明

yy/mm/dd

取得した年（西暦下 2 けた）月日（半角数字 8 文字）です。

hh:mm:ss

取得した時分秒（半角数字 8 文字）です。

AA....AA

取得日時のマイクロ秒（10 進数）です。

BB....BB

トレース取得通番（10 進数）です。

CC....CC

スレッド ID（10 進数）です。

DD....DD

コネクションハンドル（16 進数）です。

EE....EE

オブジェクトハンドル（16 進数）です。

FF....FF

セレクタ数（10 進数）です。

GG....GG
　セレクタ（10 進数）です。

HH....HH
　定数名です。

II....II
　整数型属性数（10 進数）です。

JJ....JJ
　文字型属性長（10 進数）です。

KK....KK
　完了コード（10 進数）です。

LL....LL
　理由コード（10 進数）です。

MM....MM
　整数型属性値（10 進数）です。

NN....NN
　文字型属性値（文字列）です。

出力形式（MQBEGIN の場合）

```
関数　＝　MQBEGIN（出口）
            取得日時　＝　yy/mm/dd hh:mm:ss
            マイクロ秒　＝　AA....AA
            トレース取得通番　＝　BB....BB
            スレッドID　＝　CC....CC
            コネクションハンドル　＝　[0x DD....DD]
            完了コード　＝　EE....EE（FF....FF）※1
            理由コード　＝　GG....GG（FF....FF）※1
```

注※ 1
　出口情報を表示するときだけ出力します。

● 出力項目の説明

yy/mm/dd
　取得した年（西暦下 2 けた）月日（半角数字 8 文字）です。

hh:mm:ss
　取得した時分秒（半角数字 8 文字）です。

AA....AA
　取得日時のマイクロ秒（10 進数）です。

BB....BB
　トレース取得通番（10 進数）です。

CC....CC
　スレッド ID（10 進数）です。

DD....DD
　コネクションハンドル（16 進数）です。

EE....EE
　完了コード（10 進数）です。

FF....FF
　定数名です。

GG....GG
　理由コード（10 進数）です。

出力形式（MQCMIT の場合）

```
関数　＝　MQCMIT（出口）
　　取得日時　＝　yy/mm/dd hh:mm:ss
　　マイクロ秒　＝　AA....AA
　　トレース取得通番　＝　BB....BB
　　スレッドID　＝　CC....CC
　　コネクションハンドル　＝　[0x DD....DD]
　　完了コード　＝　EE....EE（FF....FF）※1
　　理由コード　＝　GG....GG（FF....FF）※1
```

注※ 1
　出口情報を表示するときだけ出力します。

● 出力項目の説明

yy/mm/dd
　取得した年（西暦下 2 けた）月日（半角数字 8 文字）です。

hh:mm:ss
　取得した時分秒（半角数字 8 文字）です。

AA....AA
　取得日時のマイクロ秒（10 進数）です。

BB....BB
　トレース取得通番（10 進数）です。

CC....CC
　スレッド ID（10 進数）です。

DD....DD
　コネクションハンドル（16 進数）です。

EE....EE
　完了コード（10 進数）です。

FF....FF
　定数名です。

GG....GG
　理由コード（10 進数）です。

出力形式（MQBACK の場合）

```
関数　＝　MQBACK（出口）
　　取得日時　＝　yy/mm/dd hh:mm:ss
　　マイクロ秒　＝　AA....AA
　　トレース取得通番　＝　BB....BB
　　スレッドID　＝　CC....CC
　　コネクションハンドル　＝　[0x DD....DD]
　　完了コード　＝　EE....EE（FF....FF）※1
　　理由コード　＝　GG....GG（FF....FF）※1
```

注※ 1
　出口情報を表示するときだけ出力します。

● 出力項目の説明

yy/mm/dd
　取得した年（西暦下 2 けた）月日（半角数字 8 文字）です。

hh:mm:ss
　取得した時分秒（半角数字 8 文字）です。

AA....AA
　取得日時のマイクロ秒（10 進数）です。

BB....BB
　トレース取得通番（10 進数）です。

CC....CC
　スレッド ID（10 進数）です。

DD....DD
　コネクションハンドル（16 進数）です。

EE....EE
　完了コード（10 進数）です。

FF....FF
　定数名です。

GG....GG
　理由コード（10 進数）です。

出力形式（スレッド ID 一覧表）

```
スレッドID一覧
        スレッドID  ＝  AA....AA
        スレッドID  ＝  AA....AA
              :
        スレッドID  ＝  AA....AA
```

● 出力項目の説明

AA....AA
　スレッド ID（10 進数）です。

出力形式（コネクションハンドル一覧表）

```
コネクションハンドル一覧
      コネクションハンドル  ＝  [0xAA....AA] (BB....BB)
      コネクションハンドル  ＝  [0xAA....AA] (BB....BB)
            :
      コネクションハンドル  ＝  [0xAA....AA] (BB....BB)
```

● 出力項目の説明

AA....AA
　コネクションハンドル（16 進数）です。

BB....BB
　キューマネジャ名です。

4 MQC クライアント機能の MQI

この章では，C 言語または COBOL 言語の MQI を使用したクライアントアプリケーションの作成方法について説明します。

使用できる MQI
MQBACK **命令　－　ローカルトランザクションのロールバック**
MQBEGIN **命令　－　ローカルトランザクションの開始**
MQCMIT **命令　－　ローカルトランザクションのコミット**
MQBO **構造体　－　ローカルトランザクション開始オプション**
MQI **のサンプルアプリケーション**
MQI **のサンプルコーディング（**C **言語）**
MQI **のサンプルコーディング（**COBOL **言語）**

使用できる MQI

MQC クライアント機能は OpenTP1 システム以外の TM と接続または非接続で動作します。

MQC クライアント機能のクライアントアプリケーションは，TP1/Message Queue の MQI を使用できます。MQI はトランザクションの範囲およびトランザクションの範囲外で使用できます。ただし，使用する TM が X/Open の XA インタフェースに準拠している必要があります。

MQC クライアント機能のクライアントアプリケーションが使用できる MQI を次の表に示します。

TM と接続しない場合にだけ，項番 10 から 12 のローカルトランザクション機能が使用できます。

表 4-1 MQC クライアント機能のクライアントアプリケーションが使用できる MQI 一覧

項番	関数名	機能
1	MQCLOSE	オブジェクトのクローズ
2	MQCONN※	キューマネジャへの接続
3	MQDISC	キューマネジャからの切り離し
4	MQGET	メッセージの取り出し
5	MQINQ	オブジェクトの属性の照会
6	MQOPEN	オブジェクトのオープン
7	MQPUT	メッセージの登録
8	MQPUT1	1 メッセージの登録
9	MQSET	オブジェクトの属性の設定
10	MQBACK	ローカルトランザクションのロールバック
11	MQBEGIN	ローカルトランザクションの開始
12	MQCMIT	ローカルトランザクションのコミット

注※

コネクションハンドルの有効範囲は，環境のパラレル処理の最小単位までです。MQCONN 命令を呼び出したパラレル処理の単位外ではこのハンドルは無効です。TP1/Message Queue - Access のパラレル処理の最小単位はスレッドです。

MQI の詳細については，項番 1 から 9 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。項番 10 から 12 は以降のページで説明します。

MQC クライアント機能のクライアントアプリケーションのコンパイルおよびリンケージ

時に使用するライブラリファイルについては,「2.1　MQC クライアント機能のセットアップ」を参照してください。

MQBACK 命令 － ローカルトランザクションのロールバック

機能

MQBACK 命令では，最後の同期点以降に発生したメッセージの読み取りと書き込みをすべてロールバックすることをキューマネジャに指示します。トランザクションの一部として書き込まれたメッセージは削除されます。トランザクションの一部として取り出されたメッセージはキューに戻されます。

形式

C 言語の場合

```
MQBACK (MQHCONN Hconn, MQLONG *CompCode, MQLONG *Reason)
```

COBOL 言語の場合

```
CALL 'MQBACK' USING HCONN, COMPCODE, REASON.
```

引数

● Hconn（MQHCONN 型） － input

コネクションハンドルです。

キューマネジャへの接続を示すハンドルです。MQCONN 命令の戻り値を指定してください。

● CompCode（MQLONG 型） － output

完了コードです。次のどれかが返されます。

MQCC_OK：成功

MQCC_WARNING：警告（一部成功）

MQCC_FAILED：失敗

● Reason（MQLONG 型） － output

理由コードです。

表 4-2 CompCode 引数が MQCC_OK の場合（MQBACK 命令の場合）

理由コード	意味
MQRC_NONE	理由コードはありません。

表 4-3　CompCode 引数が MQCC_WARNING の場合（MQBACK 命令の場合）

理由コード	意味
MQRC_OUTCOME_PENDING	コミットまたはロールバック操作の結果が保留状態です。

表 4-4　CompCode 引数が MQCC_FAILED の場合（MQBACK 命令の場合）

理由コード	意味
MQRC_CALL_IN_PROGRESS	前の呼び出しが完了する前に，再度 MQI 呼び出しが実行されました。
MQRC_CONNECTION_BROKEN	キューマネジャとの接続が失われました。
MQRC_ENVIRONMENT_ERROR	呼び出しが環境内で有効ではありません。
MQRC_HCONN_ERROR	コネクションハンドルが無効です。
MQRC_OBJECT_DAMAGED	オブジェクトが破損しています。
MQRC_OUTCOME_MIXED	コミットまたはロールバック操作の結果が混合しています。
MQRC_Q_MGR_STOPPING	キューマネジャが終了処理中です。
MQRC_RESOURCE_PROBLEM	システム資源が不足しています。
MQRC_STORAGE_NOT_AVAILABLE	記憶容量が不足しています。
MQRC_UNEXPECTED_ERROR	予期しないエラーが発生しました。

「理由コード」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。

注意事項

● この命令は，適切な TM が存在しない環境でだけ使用できます。
このような環境では，キューマネジャ自体がトランザクションを管理します（ローカルトランザクション）。
- ローカルトランザクションおよびグローバルトランザクションの詳細については，「MQBEGIN 命令」を参照してください。
- 適切な TM が存在する環境で，トランザクションをロールバックする場合は，MQBACK 命令ではなく適切なロールバック命令を使用するか，アプリケーションを異常終了させてトランザクションをロールバックする必要があります。

● 1 トランザクションには，一つのコネクションハンドルと同じ有効範囲があります。
つまり，特定のトランザクションに影響を与える MQI 命令は，同じコネクションハンドルを使用して実行しなければなりません。例えば，別のアプリケーションで命令を呼び出すなど，別のコネクションハンドルを用いて命令を呼び出すと，別のトランザクションに影響が及びます。
コネクションハンドルの有効範囲については，マニュアル「TP1/Message Queue プログラム作成リファレンス」の MQCONN 命令を参照してください。

- この命令で影響を受けるメッセージは，現在のトランザクションの一部として書き込まれたメッセージ，または取り出されたメッセージに限られます。
- 長時間実行しているアプリケーションで，1 トランザクションに対して MQGET 命令，MQPUT 命令，または MQPUT1 命令を呼び出している場合，コミット命令またはロールバック命令を一度も呼び出さないと，ほかのアプリケーションでは使用できないメッセージで，キューがいっぱいになることがあります。
- トランザクションが発生していない状態で MQBACK 命令を発行した場合は MQRC_HCONN_ERROR のエラーになります。

MQBEGIN 命令 － ローカルトランザクションの開始

機能

MQBEGIN 命令で，キューマネジャによって管理されるトランザクションを開始します。

形式

C 言語の場合

```
MQBEGIN (MQHCONN Hconn, MQBO *BeginOptions, MQLONG *CompCode,
         MQLONG *CompReason)
```

COBOL 言語の場合

```
CALL 'MQBEGIN' USING HCONN, BEGINOPTIONS, COMPCODE, REASON.
```

引数

● Hconn（MQHCONN 型） － input

コネクションハンドルです。

キューマネジャへの接続を示すハンドルです。MQCONN 命令の戻り値を指定してください。

● BeginOptions（MQBO 構造体） － input

MQBEGIN 命令の動作を制御するオプションです。

詳細については，「MQBO 構造体」を参照してください。

BeginOptions は，予約済みパラメタです。C 言語で作成されたプログラムでは，MQBO 構造体のアドレスを指定しないで，NULL アドレスを指定できます。

● CompCode（MQLONG 型） － output

完了コードです。次のどちらかが返されます。

MQCC_OK：成功

MQCC_FAILED：失敗

● Reason（MQLONG 型） － output

理由コードです。

表 4-5 CompCode 引数が MQCC_OK の場合（MQBEGIN 命令の場合）

理由コード	意味
MQRC_NONE	理由コードはありません。

表 4-6 CompCode 引数が MQCC_FAILED の場合（MQBEGIN 命令の場合）

理由コード	意味
MQRC_BO_ERROR	開始オプション構造体が無効です。
MQRC_CALL_IN_PROGRESS	前の呼び出しが完了する前に，再度 MQI 呼び出しが実行されました。
MQRC_CONNECTION_BROKEN	キューマネジャとの接続が失われました。
MQRC_ENVIRONMENT_ERROR	呼び出しが環境内で有効ではありません。
MQRC_HCONN_ERROR	コネクションハンドルが無効です。
MQRC_OPTIONS_ERROR	オプションが有効でないか，整合性がありません。
MQRC_Q_MGR_STOPPING	キューマネジャが終了処理中です。
MQRC_RESOURCE_PROBLEM	システム資源が不足しています。
MQRC_STORAGE_NOT_AVAILABLE	記憶容量が不足しています。
MQRC_UNEXPECTED_ERROR	予期しないエラーが発生しました。
MQRC_UOW_IN_PROGRESS	トランザクションが開始済みです。

「理由コード」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。

注意事項

● MQBEGIN 命令は，キューマネジャで管理されるトランザクションの開始に使用します。
キューマネジャは，次の二つのタイプのトランザクションをサポートします。

1. キューマネジャで管理されるローカルトランザクション
参加プログラムがキューマネジャだけのトランザクションであり，キューマネジャが TM として機能します。
   - ローカルトランザクションを開始するには，トランザクション内の最初の MQPUT，MQPUT1，または MQGET 命令に MQPMO_SYNCPOINT または MQGMO_SYNCPOINT オプションを指定してください。ローカルトランザクションを開始するのに，アプリケーションが MQBEGIN 命令を呼び出す必要はありません。
   - ローカルトランザクションをコミットまたはロールバックするには，MQCMIT または MQBACK 命令を使用する必要があります。アプリケーションでどちらの命令も呼び出さない場合，このトランザクションは，MQDISC 命令を呼び出すとコ

ミットされますが，MQDISC 命令を呼び出さないで終了するとロールバックされます。

2. 外部管理されるグローバルトランザクション
キューマネジャが参加プログラムであるが，TM としては機能しないトランザクションです。その代わりに，キューマネジャが接続している TM が存在します。
- グローバルトランザクションを開始するには，TM が提供する関連命令を使用する必要があります。トランザクションを開始するために MQBEGIN 命令を使用すると失敗し，理由コード MQRC_ENVIRONMENT_ERROR が戻ります。
- グローバルトランザクションをコミットまたはロールバックするには，TM が提供するコミット命令およびロールバック命令を使用しなければなりません。
MQCMIT および MQBACK 命令は使用できません。

● アプリケーションが一度に参加プログラムとしてかかわれるトランザクションは，一つだけです。アプリケーションで MQBEGIN 命令を呼び出す場合，そのアプリケーション用のトランザクションがすでに存在していると，その命令は失敗し，理由コード MQRC_UOW_IN_PROGRESS が戻ります。

MQCMIT 命令 － ローカルトランザクションのコミット

機能

アプリケーションが同期点に達していることと，最後の同期点が永続化されてから発生した，すべてのメッセージの読み取りと書き込みをキューマネジャに指示します。

形式

C 言語の場合

```
MQCMIT (MQHCONN Hconn, MQLONG *CompCode, MQLONG *Reason)
```

COBOL 言語の場合

```
CALL 'MQCMIT' USING HCONN, COMPCODE, REASON.
```

引数

● Hconn（MQHCONN 型） － input

コネクションハンドルです。

キューマネジャへの接続を示すハンドルです。MQCONN 命令の戻り値を指定してください。

● CompCode（MQLONG 型） － output

完了コードです。

次のどれかが返されます。

MQCC_OK：成功

MQCC_WARNING：警告（一部成功）

MQCC_FAILED：失敗

● Reason（MQLONG 型） － output

理由コードです。

表 4-7 CompCode 引数が MQCC_OK の場合（MQCMIT 命令の場合）

理由コード	意味
MQRC_NONE	理由コードはありません。

表 4-8 CompCode 引数が MQCC_WARNING の場合（MQCMIT 命令の場合）

理由コード	意味
MQRC_BACKED_OUT	同期点は完了しましたが，資源がロールバックしました。
MQRC_OUTCOME_PENDING	コミットまたはロールバック操作の結果が保留状態です。

表 4-9 CompCode 引数が MQCC_FAILED の場合（MQCMIT 命令の場合）

理由コード	意味
MQRC_CALL_IN_PROGRESS	前の呼び出しが完了する前に，再度 MQI 呼び出しが実行されました。
MQRC_CONNECTION_BROKEN	キューマネジャとの接続が失われました。
MQRC_ENVIRONMENT_ERROR	呼び出しが環境内で有効ではありません。
MQRC_HCONN_ERROR	コネクションハンドルが無効です。
MQRC_OBJECT_DAMAGED	オブジェクトが破損しています。
MQRC_OUTCOME_MIXED	コミットまたはロールバック操作の結果が混合しています。
MQRC_Q_MGR_STOPPING	キューマネジャが終了処理中です。
MQRC_RESOURCE_PROBLEM	システム資源が不足しています。
MQRC_STORAGE_NOT_AVAILABLE	記憶容量が不足しています。
MQRC_UNEXPECTED_ERROR	予期しないエラーが発生しました。

「理由コード」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。

注意事項

● この命令は，適切な TM が存在しない環境だけで使用できます。
このような環境では，キューマネジャ自体がトランザクションを管理します（ローカルトランザクション）。
- ローカルトランザクションおよびグローバルトランザクションの詳細については，「MQBEGIN 命令」を参照してください。
- 適切な TM が存在する環境で，トランザクションをコミットする場合は，MQCMIT 命令ではなく適切なコミット命令を使用してトランザクションをコミットする必要があります。

● 1 トランザクションには，一つのコネクションハンドルと同じ有効範囲があります。つまり，特定のトランザクションに影響を与える MQI 命令は，同じコネクションハンドルを使用して実行しなければなりません。例えば，別のアプリケーションで命令を呼び出すなど，別のコネクションハンドルを用いて命令を発行すると，別のトランザクションに影響が及びます。

コネクションハンドルの有効範囲については，マニュアル「TP1/Message Queue プログラム作成リファレンス」の MQCONN 命令を参照してください。

- この命令で影響を受けるメッセージは，現行のトランザクションの一部として書き込まれたメッセージ，または取り出されたメッセージに限られます。
- トランザクション内部にコミットされていない変更内容がある場合，アプリケーションが MQCMIT 命令および MQBACK 命令を呼び出さないで終了すると，それらの変更内容に対する後処理は，そのアプリケーションの終了方法によって異なります。
  - アプリケーションが終了前に MQDISC 命令を呼び出す場合，その命令によってトランザクションがコミットされます。
  - アプリケーションが異常終了する場合，または MQDISC 命令を呼び出さない場合，トランザクションはロールバックされます。これは，MQBACK 命令が呼び出されるのと同様です。
- 長時間実行しているアプリケーションで，1 トランザクションに対して MQGET 命令，MQPUT 命令，または MQPUT1 命令を呼び出している場合，コミット命令またはロールバック命令を一度も呼び出さないと，キューがほかのアプリケーションでは使用できないメッセージで満たされることがあります。
- Reason パラメタが MQRC_CONNECTION_BROKEN で，CompCode パラメタが MQCC_FAILED の場合，トランザクションは正常にコミットされた可能性があります。
- トランザクションが発生していない状態で MQCMIT 命令を発行した場合は MQRC_HCONN_ERROR のエラーになります。

MQBO 構造体　－　ローカルトランザクション開始オプション

MQBO 構造体は，MQBEGIN 命令で使用する入力用の引数です。次のフィールドから構成されます。

表 4-10　MQBO 構造体を構成するフィールド一覧

フィールド（データタイプ）	内容	初期値
StrucId（MQCHAR4 型）	構造体識別子	MQBO_STRUC_ID
Version（MQLONG 型）	構造体バージョン番号	MQBO_VERSION_1
Options（MQLONG 型）	開始オプション	MQBO_NONE

フィールド

●　StructId（MQCHAR4 型）　構造体識別子

次の値をとります。

MQBO_STRUC_ID：ローカルトランザクション開始オプションの構造体識別子

C 言語では，MQBO_STRUC_ID_ARRAY も定義されています。これは，MQBO_STRUC_ID と同じ値です。ただし，文字列ではなく文字の配列として定義されています。

これは，入力用のフィールドです。

このフィールドの初期値は，MQBO_STRUC_ID です。

●　Version（MQLONG 型）　構造体バージョン番号

次の値をとります。

MQBO_VERSION_1：ローカルトランザクション開始オプションの構造体バージョン番号

これは，入力用のフィールドです。

このフィールドの初期値は，MQBO_VERSION_1 です。

●　Options（MQLONG 型）　開始オプション

MQBEGIN 命令の動作を制御します。

指定できるオプションを説明します。

MQBO_NONE：オプションを指定しません。

このオプションは，ほかにオプションを指定しない場合に指定します。

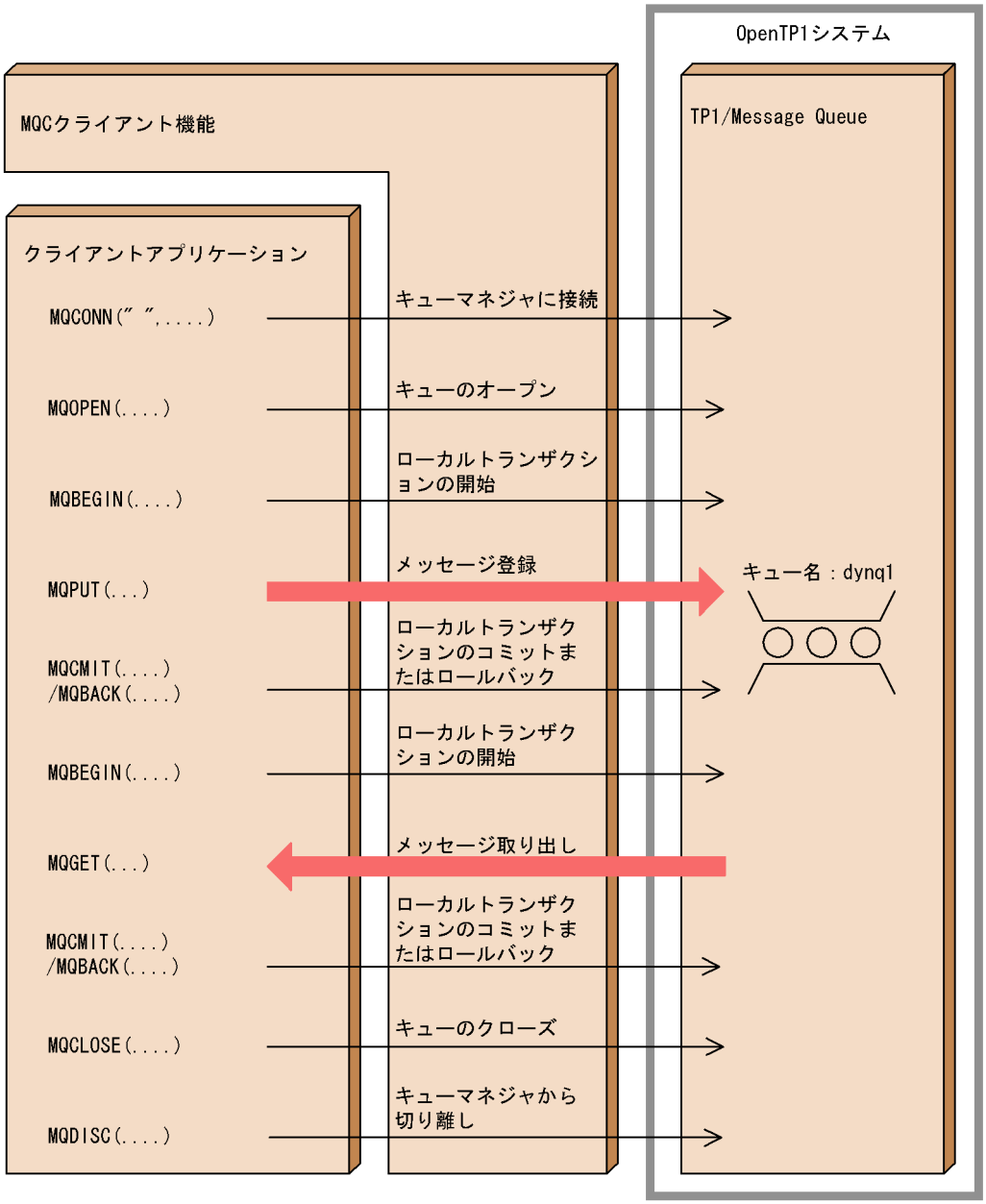
これは，入力用のフィールドです。

このフィールドの初期値は，MQBO_NONE です。

MQI のサンプルアプリケーション

ローカルトランザクションを使用する場合の処理の流れを次の図に示します。

図 4-1　処理の流れ



クライアントアプリケーション作成時の注意（MQI）

- トランザクションが中断している間（xa_end(TM_SUSPEND)）は，別のトランザクションを開始できません。
- クライアントアプリケーションをマルチスレッドで作成する（環境変数に DCMQCEXPTRN=Y を設定している）場合，MQCONN 命令はスレッドごとに発行してください。MQCONN 命令で取得したコネクションハンドルは異なるスレッドでは使用できません。

MQI のサンプルコーディング（C 言語）

C 言語でのコーディング例を示します。

```
/* All Rights Reserved, Copyright (C) 2003, Hitachi, Ltd. */
/************************************************************
**  mqcsample.c
**  functions = main()
************************************************************/

/************************************************************
**  name = main()
**  func = main function of UAP
**       (1)start UAP(output start message)
**       (2)MQCONN(Connect queue manager)
**       (3)MQOPEN(Open queue)
**       (4)MQBEGIN(begin local transaction)
**       (5)MQPUT(Put message)
**       (6)MQBACK(rollback local transaction)
**       (7)MQCMIT(commit local transaction)
**       (8)MQBEGIN(begin local transaction)
**       (9)MQGET(Get message)
**       (10)MQBACK(rollback local transaction)
**       (11)MQCMIT(commit local transaction)
**       (12)MQCLOSE(Close queue)
**       (13)MQDISC(Disconnect queue manager)
**       (14)finish UAP(output end message)
************************************************************/
#include    <stdio.h>
#include    <string.h>
#include    <cmqc.h>

#define PUT_DATA     "******** sample put data ********"

int main(void)
{
    MQCHAR48    QueueManager = " ";       /* queue manager name  */
    MQLONG      comp_code,                /* completion code     */
                reason;                   /* reason code         */
    MQHCONN     qm_handle;                /* connection handle   */
    MQCHAR      message_data[200];        /* buffer              */
    MQHOBJ      que_handle;               /* object handle       */
    MQLONG      buffer_length;            /* buffer length       */
    MQLONG      data_length;              /* message length      */
    MQLONG      open_options;             /* open options        */
    MQOD        obj_desc = { MQOD_DEFAULT };
                                          /* object descriptor   */
    MQMD        msg_desc = { MQMD_DEFAULT };
                                          /* message descriptor  */
    MQPMO       put_options = { MQPMO_DEFAULT };
                                          /* put message options */
    MQGMO       get_options = { MQGMO_DEFAULT };
                                          /* get message options */
    MQBO        begin_options = { MQBO_DEFAULT };
                                          /* begin options       */
```

```
    /* set the queue name */
    strcpy(obj_desc.ObjectName, "dynq1");

    /* set the message data */
    strcpy(message_data, PUT_DATA) ;
    buffer_length = strlen(message_data);

    printf("MQTRN:Start service. ¥n");

    /* MQCONN( connect to the queue manager ) */
    MQCONN(QueueManager, &qm_handle, &comp_code, &reason);
    if (comp_code != MQCC_OK)
    {
        printf("MQTRN:Failed at MQCONN. CODE = %ld¥n", reason);
        goto PROG_END;
    }

    /* MQOPEN( open the queue ) */
    open_options = MQOO_OUTPUT | MQOO_INPUT_AS_Q_DEF;
    MQOPEN(qm_handle, &obj_desc, open_options, &que_handle,
            &comp_code, &reason);
    if (comp_code != MQCC_OK)
    {
        printf("MQTRN:Failed at MQOPEN. CODE = %ld¥n", reason);
        goto MQ_END;
    }

    /* MQBEGIN( begin local transaction ) */
    MQBEGIN(qm_handle, &begin_options, &comp_code, &reason);
    if (comp_code != MQCC_OK)
    {
        printf("MQTRN:Failed at MQBEGIN. CODE = %ld¥n",reason);
        goto MQ_END;
    }

    /* MQPUT( put the message ) */
    put_options.Options = MQPMO_SYNCPOINT;
    MQPUT(qm_handle, que_handle, &msg_desc, &put_options,
            buffer_length, (PMQBYTE)message_data,&comp_code,
&reason);
    if (comp_code != MQCC_OK)
    {
        printf("MQTRN:Failed at MQPUT. CODE = %ld¥n", reason);
        /* MQBACK( rollback local transaction ) */
        MQBACK(qm_handle, &comp_code, &reason);
        if (comp_code != MQCC_OK)
        {
            printf("MQTRN:Failed at MQBACK. CODE = %ld¥n",reason);
        }
        goto MQ_END;
    }

    /* MQCMIT( commit local transaction ) */
    MQCMIT(qm_handle, &comp_code, &reason);
    if (comp_code != MQCC_OK)
    {
        printf("MQTRN:Failed at MQCMIT. CODE = %ld¥n", reason);
```

```
        goto MQ_END;
    }

    /* MQBEGIN( begin local transaction ) */
    MQBEGIN(qm_handle, &begin_options, &comp_code, &reason);
    if (comp_code != MQCC_OK)
    {
        printf("MQTRN:Failed at MQBEGIN. CODE = %ld¥n",reason);
        goto MQ_END;
    }

    /* MQGET( get the message ) */
    get_options.Options = MQGMO_SYNCPOINT | MQGMO_NO_WAIT;
    MQGET(qm_handle, que_handle, &msg_desc, &get_options,
            buffer_length, (PMQBYTE)message_data,
            &data_length, &comp_code, &reason);
    if (comp_code != MQCC_OK)
    {
        printf("MQTRN:Failed at MQGET. CODE = %ld¥n", reason);
        /* MQBACK( rollback local transaction ) */
        MQBACK(qm_handle, &comp_code, &reason);
        if (comp_code != MQCC_OK)
        {
            printf("MQTRN:Failed at MQBACK. CODE = %ld¥n",reason);
        }
        goto MQ_END;
    }

    /* MQCMIT( commit local transaction ) */
    MQCMIT(qm_handle, &comp_code, &reason);
    if (comp_code != MQCC_OK)
    {
        printf("MQTRN:Failed at MQCMIT. CODE = %ld¥n", reason);
        goto MQ_END;
    }

    /* MQCLOSE( close the queue ) */
    MQCLOSE(qm_handle, &que_handle, MQCO_NONE, &comp_code,
                &reason);
    if (comp_code != MQCC_OK)
    {
        printf("MQTRN:Failed at MQCLOSE. CODE = %ld¥n",reason);
    }

MQ_END:
    /* MQDISC( disconnect from the queue manager ) */
    MQDISC(&qm_handle, &comp_code, &reason);
    if (comp_code != MQCC_OK)
    {
        printf("MQTRN:Failed at MQDISC. CODE = %ld¥n", reason);
    }

PROG_END:
    printf("MQTRN:Terminate service. ¥n");
    return(0);
}
```

MQI のサンプルコーディング（COBOL 言語）

COBOL 言語でのコーディング例を示します。

```
*******************************************************
*        MQ-ACCESS SAMPLE                             *
*******************************************************
*
 IDENTIFICATION DIVISION.
*
 PROGRAM-ID. MAIN.
*
*******************************************************
*        INITIALIZE                                   *
*******************************************************
*
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*
 01  MQ-MGRNAME          PIC X(48) VALUE SPACES.
 01  MQ-HCONN            PIC S9(9) BINARY.
 01  MQ-COMPCODE         PIC S9(9) BINARY.
 01  MQ-REASON           PIC S9(9) BINARY.
 01  MQ-HOBJ             PIC S9(9) BINARY.
 01  MQ-OPTIONS          PIC S9(9) BINARY.
 01  MQ-BUFFLEN          PIC S9(9) BINARY.
 01  MQ-DATALEN          PIC S9(9) BINARY.
 01  MQ-BUFFER           PIC X(200) VALUE SPACE.
*
 01  PUT-DATA            PIC X(33) VALUE
                  '***** COBOL sample put data *****'.
 01  QUEUENAME           PIC X(48) VALUE
                         'dynq1    '.
*
 01 MQ-OBJECT-DESC.
    COPY CMQODV.
*
 01 MQ-MESSAGE-DESCRIPTOR.
    COPY CMQMDV.
*
 01 MQ-PUT-MESSAGE-OPTIONS.
    COPY CMQPMOV.
*
 01 MQ-GET-MESSAGE-OPTIONS.
    COPY CMQGMOV.
*
 01 MQ-BEGIN-OPTIONS.
    COPY CMQBOV.
*
 01 MQ-CONSTANTS.
    COPY CMQV SUPPRESS.
*
 PROCEDURE DIVISION.
*
```

```
*******************************************************
*        CONNECT TO THE QUEUE MANAGER                 *
*******************************************************
*
 CALL 'MQCONN' USING MQ-MGRNAME
                     MQ-HCONN
                     MQ-COMPCODE
                     MQ-REASON.
     IF MQ-COMPCODE NOT = MQCC-OK THEN
       DISPLAY 'MQTRN:MQCONN FAILED. REASON CODE = '
       MQ-REASON
       STOP RUN
     END-IF.
*
*******************************************************
*        OPEN THE QUEUE                               *
*******************************************************
*
 MOVE QUEUENAME TO MQOD-OBJECTNAME.
*
 COMPUTE MQ-OPTIONS = MQOO-OUTPUT +
                                  MQOO-INPUT-AS-Q-DEF.
*
 CALL 'MQOPEN' USING MQ-HCONN
                     MQOD
                     MQ-OPTIONS
                     MQ-HOBJ
                     MQ-COMPCODE
                     MQ-REASON.
     IF MQ-COMPCODE NOT = MQCC-OK THEN
       DISPLAY 'MQTRN:MQOPEN FAILED. REASON CODE = '
       MQ-REASON
       GO TO PROG-END
     END-IF.
*
*******************************************************
*        SET THE PUT MESSAGE OPTIONS                  *
*******************************************************
*
 MOVE MQPMO-SYNCPOINT     TO MQPMO-OPTIONS.
 MOVE 33                  TO MQ-BUFFLEN.
*
 MOVE PUT-DATA            TO MQ-BUFFER.
*
*******************************************************
*        SET THE GET MESSAGE OPTIONS                  *
*******************************************************
*
 MOVE MQGMO-SYNCPOINT     TO MQGMO-OPTIONS.
*
*******************************************************
*        BEGIN LOCAL TRANSACTION                      *
*******************************************************
*
 CALL 'MQBEGIN' USING MQ-HCONN
                      MQ-BEGIN-OPTIONS
                      MQ-COMPCODE
```

```
                        MQ-REASON.
      IF MQ-COMPCODE NOT = MQCC-OK THEN
        DISPLAY 'MQTRN:MQBEGIN FAILED. REASON CODE = '
        MQ-REASON
        STOP RUN
      END-IF.
*
*****************************************************
*         PUT THE MESSAGE                           *
*****************************************************
*
 CALL 'MQPUT' USING MQ-HCONN
                    MQ-HOBJ
                    MQMD
                    MQPMO
                    MQ-BUFFLEN
                    MQ-BUFFER
                    MQ-COMPCODE
                    MQ-REASON.
      IF MQ-COMPCODE NOT = MQCC-OK THEN
        DISPLAY 'MQTRN:MQPUT FAILED. REASON CODE = '
        MQ-REASON
      END-IF.
*
*****************************************************
*         COMMIT LOCAL TRANSACTION                  *
*****************************************************
*
 CALL 'MQCMIT' USING MQ-HCONN
                     MQ-COMPCODE
                     MQ-REASON.
      IF MQ-COMPCODE NOT = MQCC-OK THEN
        DISPLAY 'MQTRN:MQCMIT FAILED. REASON CODE = '
        MQ-REASON  MQ-COMPCODE
        STOP RUN
      END-IF.
*
*****************************************************
*         BEGIN LOCAL TRANSACTION                   *
*****************************************************
*
 CALL 'MQBEGIN' USING MQ-HCONN
                      MQ-BEGIN-OPTIONS
                      MQ-COMPCODE
                      MQ-REASON.
      IF MQ-COMPCODE NOT = MQCC-OK THEN
        DISPLAY 'MQTRN:MQBEGIN FAILED. REASON CODE = '
        MQ-REASON
        STOP RUN
      END-IF.
*
*****************************************************
*         GET THE MESSAGE                           *
*****************************************************
*
 CALL 'MQGET' USING MQ-HCONN
                    MQ-HOBJ
```

```
                    MQMD
                    MQGMO
                    MQ-BUFFLEN
                    MQ-BUFFER
                    MQ-DATALEN
                    MQ-COMPCODE
                    MQ-REASON.
     IF MQ-COMPCODE NOT = MQCC-OK THEN
       DISPLAY 'MQTRN:MQGET FAILED. REASON CODE = '
       MQ-REASON
     END-IF.
*
******************************************************
*         COMMIT LOCAL TRANSACTION                   *
******************************************************
*
 CALL 'MQCMIT' USING MQ-HCONN
                     MQ-COMPCODE
                     MQ-REASON.
     IF MQ-COMPCODE NOT = MQCC-OK THEN
       DISPLAY 'MQTRN:MQCMIT FAILED. REASON CODE = '
       MQ-REASON  MQ-COMPCODE
       STOP RUN
     END-IF.
*
******************************************************
*         CLOSE THE QUEUE                            *
******************************************************
*
 MOVE MQCO-NONE TO MQ-OPTIONS.
*
 CALL 'MQCLOSE' USING MQ-HCONN
                      MQ-HOBJ
                      MQ-OPTIONS
                      MQ-COMPCODE
                      MQ-REASON.
     IF MQ-COMPCODE NOT = MQCC-OK THEN
       DISPLAY 'MQTRN:MQCLOSE FAILED. REASON CODE = '
       MQ-REASON
     END-IF.
*
 PROG-END.
*
******************************************************
*         DISCONNECT FROM THE QUEUE MANAGER          *
******************************************************
*
 CALL 'MQDISC' USING MQ-HCONN
                     MQ-COMPCODE
                     MQ-REASON.
     IF MQ-COMPCODE NOT = MQCC-OK THEN
       DISPLAY 'MQTRN:MQDISC FAILED. REASON CODE = '
       MQ-REASON
     END-IF.
*
 STOP RUN.
 END PROGRAM MAIN.
```

5 MQC クライアント機能の C++ インタフェース

この章では，MQC クライアント機能の C++ インタフェースについて説明します。

MQC クライアント機能の C++ クラス一覧

C++ クラス継承図

ImqBinary クラス（C++）

ImqCache クラス（C++）

ImqDeadLetterHeader クラス（C++）

ImqDistributionList クラス（C++）

ImqError クラス（C++）

ImqGetMessageOptions クラス（C++）

ImqHeader クラス（C++）

ImqItem クラス（C++）

ImqMessage クラス（C++）

ImqMessageTracker クラス（C++）

ImqObject クラス（C++）

ImqProcess クラス（C++）

ImqPutMessageOptions クラス（C++）

ImqQueue クラス（C++）

ImqQueueManager クラス（C++）

ImqReferenceHeader クラス（C++）

ImqString クラス（C++）

ImqTrigger クラス（C++）

C++ のサンプルアプリケーション

C++ のサンプルコーディング

MQC クライアント機能の C++ クラス一覧

MQC クライアント機能が提供する C++ クラスの一覧を次の表に示します。

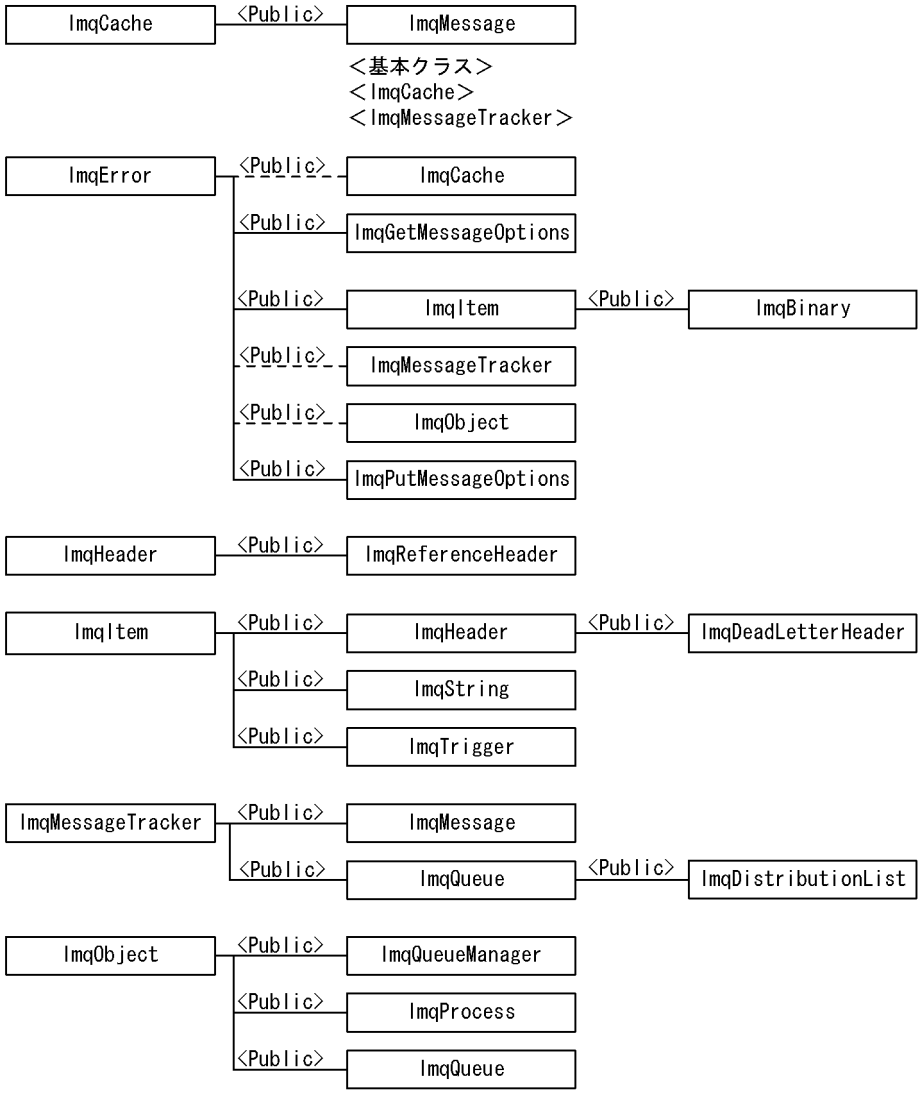
表 5-1 クラスの一覧（C++）

クラス名	機能
ImqBinary	2 進バイトアレイをカプセル化します。
ImqCache	データのメモリ保持および整理をカプセル化します。
ImqDeadLetterHeader	MQDLH 構造体の特定の機能をカプセル化します。
ImqDistributionList	配布リストをカプセル化します。
ImqError	エラーに関する情報を提供します。
ImqGetMessageOptions	MQGMO 構造体をカプセル化します。
ImqHeader	MQDLH 構造体の共通機能をカプセル化します。
ImqItem	一つのメッセージ内の一つの項目です。
ImqMessage	MQMD 構造体のカプセル化します。
ImqMessageTracker	ImqMessage または ImqQueue オブジェクトと関連づけできる属性をカプセル化します。
ImqObject	キューマネジャおよびキューの属性をカプセル化します。
ImqProcess	トリガモニタによってトリガできるアプリケーションプロセスの属性をカプセル化します。
ImqPutMessageOptions	MQPMO 構造体をカプセル化します。
ImqQueue	キューの属性およびアクセスをカプセル化します。
ImqQueueManager	キューマネジャの属性およびアクセスをカプセル化します。
ImqReferenceHeader	MQRMH 構造体をカプセル化します。
ImqString	ヌル文字で終了する文字列に，記憶域と操作を提供します。
ImqTrigger	MQTM 構造体をカプセル化します。

C++ クラス継承図

クラス継承図を次の図に示します。

図 5-1 クラス継承図（C++ の場合）



（凡例）
- - - - - - - - - - ：仮想継承します。
―――――――― ：継承します。

ImqBinary クラス（C++）

このクラスは，ImqMessageTracker クラスの accounting token，correlation id，および message id 値に使用できる 2 進バイトアレイをカプセル化します。これによって，割り当て，コピー，および比較が容易になります。

インクルードファイル

このクラスを使用するときは，imqbin.hpp ファイルをインクルードしてください。

オブジェクト属性

- data

2 進データのバイトアレイです。初期値はヌル文字です。

- data length

バイト数です。初期値はヌルです。

- data pointer

data の最初のバイトのアドレスです。初期値はヌルです。

コンストラクタ

- ImqBinary( );

デフォルトのコンストラクタです。

- ImqBinary(const ImqBinary & binary);

コピーコンストラクタです。

- ImqBinary(const void * data，const size_t length);

data から length 分のバイトをコピーします。

メソッド

- virtual ImqBoolean copyOut(ImqMessage & msg);

多重定義された ImqItem メソッドです。data をメッセージバッファにコピーし，既存の内容があれば置き換えます。msg format を MQFMT_NONE に設定します。

詳細については，「ImqItem クラス（C++）」の「メソッド」を参照してください。

- virtual ImqBoolean pasteIn(ImqMessage & msg);

多重定義された ImqItem メソッドです。メッセージバッファから残っているデータを転送し，既存の data を置き換えることによって，data を設定します。正常に実行される

ためには，ImqMessage format が MQFMT_NONE でなければなりません。

詳細については，「ImqItem クラス（C++）」の「メソッド」を参照してください。

- void operator = (const ImqBinary & binary);

binary 分のバイトをコピーします。

- ImqBoolean operator == (const ImqBinary & binary);

このオブジェクトを binary と比較します。これらが等しくない場合は 0 を，それ以外の場合は「0 でない」を返します。二つのオブジェクトが同じ datalength を持っており，バイト数が一致する場合，二つのオブジェクトは等しいと見なされます。

- ImqBoolean copyOut(void * buffer，const size_t length，const char pad = 0);

最大 length 分のバイトを data pointer から buffer へコピーします。data length が不十分であるときは，buffer 内の残りのスペースには pad 分のバイトが埋められます。length も 0 である場合は，buffer が 0 であると考えられます。length が負であってはなりません。正常に実行された場合は TRUE を返します。

- size_t dataLength( ) const ;

data length を返します。

- ImqBoolean setDataLength(const size_t length);

data length を設定します。このメソッドの結果として data length が変更された場合，オブジェクト内のデータの初期設定は解除されます。正常に実行された場合は TRUE を返します。

- void * dataPointer( ) const ;

data pointer を返します。

- ImqBoolean isNull( ) const ;

data length が 0 である場合，または data バイトがすべて 0 である場合は TRUE を返します。それ以外の場合は FALSE を返します。

- ImqBoolean set(const void * buffer，const size_t length);

buffer から length 分のバイトをコピーします。正常に実行された場合は TRUE を返します。

メソッド（protected）

- void clear( );

data length を減らして 0 にします。

その他の関連クラス

ImqItem

ImqMessage

理由コード

- MQRC_NO_BUFFER
- MQRC_STORAGE_NOT_AVAILABLE

理由コードの詳細については，「付録 A　理由コード」，またはマニュアル「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。

ImqCache クラス（C++）

このクラスは，データをメモリに保持したり，整理したりするのに使用します。ユーザが固定サイズのメモリのバッファを指定したり，システムによる自動的で柔軟なメモリ設定に任せたりもできます。

インクルードファイル

このクラスを使用するときは，imqcac.hpp ファイルをインクルードしてください。

オブジェクト属性

- automatic buffer

バッファメモリがシステムによって自動的に管理されるときは TRUE，ユーザによって提供されるときは FALSE を示します。初期値は TRUE です。

- buffer length

バッファ内のメモリのバイト数です。初期値は 0 です。

- buffer pointer

バッファメモリのアドレスです。初期値はヌルです。

- data length

data pointer よりあとのバイト数です。message length に等しいか，それより小さい数値です。初期値は 0 です。

- data offset

data pointer より前のバイト数です。message length に等しいか，それより小さい数値です。初期値は 0 です。

- data pointer

読み取りまたは書き込みが実行されるバッファのアドレスです。初期値はヌルです。

- message length

バッファ内の有効なデータのバイト数です。初期値は 0 です。

コンストラクタ

- ImqCache( );

デフォルトのコンストラクタです。

- ImqCache(const ImqCache & cache);

コピーコンストラクタです。

メソッド

- void operator = (const ImqCache & cache);

message length 分のバイトのデータを cache オブジェクトから該当するオブジェクトへコピーします。automatic buffer が FALSE である場合は，buffer length が，コピーされたデータを収容できるだけの大きさでなければなりません。

- ImqBoolean automaticBuffer( ) const ;

automatic buffer の値を返します。

- size_t bufferLength( ) const ;

buffer length を返します。

- char * bufferPointer( ) const ;

buffer pointer を返します。

- void clearMessage( );

message length と data offset を両方とも 0 に設定します。

- size_t dataLength( ) const ;

data length を返します。

- size_t dataOffset( ) const ;

data offset を返します。

- ImqBoolean setDataOffset(const size_t offset);

data offset を設定します。message length が data offset 以上であることが必要なときは，message length を増やします。正常に実行された場合は TRUE を返します。

- char * dataPointer( ) const ;

data pointer のコピーを返します。

- size_t messageLength( ) const ;

message length を返します。

- ImqBoolean setMessageLength(const size_t length);

message length を設定します。

message length が buffer length 以下であることが必要なときは，buffer length を増やします。data offset が message length 以下であることが必要なときは，data offset を減らします。正常に実行された場合は TRUE を返します。

- ImqBoolean moreBytes(const size_t bytes-required);

data pointer からバッファの終わりまでの間で，さらに bytes-required 分のバイトを書き込みに使用できるようにします。正常に実行された場合は TRUE を返します。

automatic buffer が TRUE である場合は，必要に応じて，さらに多くのメモリが取得されます。それ以外の場合は，buffer length が十分な大きさでなければなりません。

- ImqBoolean read(const size_t length，char * & external-buffer);

length 分のバイトを，data pointer 位置で始まるバッファから external-buffer へコピーします。データがコピーされたあとで，data offset は length だけ増やされます。正常に実行された場合は TRUE を返します。

- ImqBoolean resizeBuffer(const size_t length);

automatic buffer が TRUE である場合は，バッファメモリを再度割り振ることによって，buffer length を変更します。

既存のバッファから最大 message length 分のバイトのデータが新しいバッファにコピーされます。コピーされる最大数は length バイトです。buffer pointer は変更されます。message length と data offset は，新しいバッファの範囲内で可能な限り接近して予約されます。正常に実行された場合は TRUE を返します。automatic buffer が FALSE である場合は FALSE を返します。

注意

システム資源に問題があると，このメソッドは失敗し，理由コード MQRC_STORAGE_NOT_AVAILABLE が返されます。

- ImqBoolean useEmptyBuffer(const char * external-buffer，const size_t length);

空のユーザバッファを識別し，buffer pointer が external-buffer を示すように設定し，buffer length を length に，また，message length を 0 にそれぞれ設定します。clearMessage を実行します。バッファがデータで完全に満たされている場合は，代わりに useFullBuffer メソッドを使用してください。バッファの一部にデータが入っている場合は，setMessageLength メソッドを使用して，正しい量を示してください。正常に実行された場合は TRUE を返します。

このように，このメソッドを使用して，固定量のメモリ（external-buffer がヌル文字でなく，length が 0 でない）を識別できます。この場合，automatic buffer は FALSE に設定されます。また，システムによる柔軟なメモリ管理（external-buffer がヌル文字で，かつ length が 0 である）に戻すこともできます。この場合，automatic buffer は TRUE

に設定されます。

● ImqBoolean useFullBuffer(const char * externalBuffer，const size_t length);

このメソッドと ImqBoolean useEmptyBuffer メソッドの違いは，message length が length に設定されていることだけです。正常に実行された場合は TRUE を返します。

● ImqBoolean write(const size_t length，const char * external-buffer);

length 分のバイトを，external-buffer から，data1 pointer 位置で始まるバッファへコピーします。データがコピーされたあとで，data offset は length だけ増やされます。また，message length が新しい data offset 値以上になるようにする必要がある場合は message length が増やされます。正常に実行された場合は TRUE を返します。

automatic buffer が TRUE である場合は，十分な量のメモリが保証されます。それ以外の場合は，最終的な data offset が buffer length を超えてはいけません。

その他の関連クラス

ImqError

理由コード

- MQRC_BUFFER_NOT_AUTOMATIC
- MQRC_DATA_TRUNCATED
- MQRC_INSUFFICIENT_BUFFER
- MQRC_INSUFFICIENT_DATA
- MQRC_NULL_POINTER
- MQRC_STORAGE_NOT_AVAILABLE
- MQRC_ZERO_LENGTH

理由コードの詳細については，「付録 A　理由コード」，またはマニュアル「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。

ImqDeadLetterHeader クラス（C++）

このクラスは，MQDLH 構造体の特定の機能をカプセル化します。このクラスのオブジェクトは，一般的に，処理不能なメッセージを検出するアプリケーションによって使用されます。送達不能ヘッダと処理不能なメッセージ内容とで構成される新規メッセージは送達不能キューに入れられ，その処理不能メッセージは破棄されます。

インクルードファイル

このクラスを使用するときは，imqdlh.hpp ファイルをインクルードしてください。

オブジェクト属性

- dead-letter reason code

メッセージが送達不能キューに届いた理由です。初期値は MQRC_NONE です。

MQDLH 構造体の，Reason フィールドに相当します。

- destination queue manager name

元の宛先キューマネジャの名前です。初期値はヌル文字です。

MQDLH 構造体の，DestQMgrName フィールドに相当します。

- destination queue name

元の宛先キューの名前です。初期値はヌル文字です。

MQDLH 構造体の，DestQName フィールドに相当します。

- put application name

メッセージを送達不能キューに書き込んだアプリケーションの名前です。初期値はヌル文字です。

MQDLH 構造体の，PutApplName フィールドに相当します。

- put application type

メッセージを送達不能キューに書き込んだアプリケーションのタイプです。初期値は 0 です。

MQDLH 構造体の，PutApplType フィールドに相当します。

- put date

メッセージが送達不能キューに書き込まれた日付です。初期値は空白です。

MQDLH 構造体の，PutDate フィールドに相当します。

- put time

メッセージが送達不能キューに書き込まれた時刻です。初期値は空白です。

MQDLH 構造体の，PutTime フィールドに相当します。

コンストラクタ

- ImqDeadLetterHeader( );

デフォルトのコンストラクタです。

- ImqDeadLetterHeader(const ImqDeadLetterHeader & header);

コピーコンストラクタです。

メソッド

- virtual ImqBoolean copyOut(ImqMessage & msg);

多重定義された ImqItem メソッドです。MQDLH 構造体をメッセージバッファの初めに挿入し，既存のメッセージデータを後ろにずらします。msg format を MQFMT_DEAD_LETTER_HEADER に設定します。

詳細については，「ImqHeader クラス（C++）」の「メソッド」を参照してください。

- virtual ImqBoolean pasteIn(ImqMessage & msg);

多重定義された ImqItem メソッドです。メッセージバッファから MQDLH 構造体を読み取ります。

正常に実行されるためには，ImqMessage format が MQFMT_DEAD_LETTER_HEADER でなければなりません。

詳細については，「ImqHeader クラス（C++）」の「メソッド」を参照してください。

- void operator = (const ImqDeadLetterHeader & header);

インスタンスデータが header からコピーされ，既存のインスタンスデータと置き換えられます。

- MQLONG deadLetterReasonCode( ) const ;

dead-letter reason code を返します。

- void setDeadLetterReasonCode(const MQLONG reason);

dead-letter reason code を設定します。

- ImqString destinationQueueManagerName( ) const ;

destination queue manager name を返します。

- void setDestinationQueueManagerName(const char * name);

destination queue manager name を設定します。

- ImqString destinationQueueName( ) const ;

destination queue name のコピーを返します。

- void setDestinationQueueName(const char * name);

destination queue name を設定します。

- ImqString putApplicationName( ) const ;

put application name のコピーを返します。

- void setPutApplicationName(const char * name = 0);

put application name を設定します。

- MQLONG putApplicationType( ) const ;

put application type を返します。

- void setPutApplicationType(const MQLONG type = MQAT_NO_CONTEXT);

put application type を設定します。

- ImqString putDate( ) const ;

put date のコピーを返します。

- void setPutDate(const char * date = 0);

put date を設定します。

- ImqString putTime( ) const ;

put time のコピーを返します。

- void setPutTime(const char * time = 0);

put time を設定します。

オブジェクト属性（protected）

- MQDLH omqdlh

MQDLH 構造体です。

その他の関連クラス

ImqHeader

ImqItem

ImqMessage

ImqString

ImqDistributionList クラス（C++）

このクラスは，複数の宛先に一つまたは複数のメッセージを送信するために，一つまたは複数のキューを参照する動的配布リストをカプセル化します。

インクルードファイル

このクラスを使用するときは，imqdst.hpp ファイルをインクルードしてください。

オブジェクト属性

- first distributed queue

ImqQueue クラスの一つまたは複数のオブジェクトの，ImqQueue distribution list reference がオブジェクトをアドレス指定する最初のキューです。初期値は 0 です。

ImqDistributionList オブジェクトがオープンされると，それを参照する ImqQueue オブジェクトはすべて自動的にクローズされます。

コンストラクタ

- ImqDistributionList();

デフォルトのコンストラクタです。

- ImqDistributionList(const ImqDistributionList & list);

コピーコンストラクタです。

メソッド

- void operator = (const ImqDistributionList & list);

このオブジェクトを参照するすべての ImqQueue オブジェクトは，コピーの前に参照の対象から外されます。このメソッドが呼び出されたあとに，このオブジェクトを参照する ImqQueue オブジェクトはありません。

- ImqQueue * firstDistributedQueue() const;

first distributed queue を返します。

メソッド（protected）

- void setFirstDistributedQueue(ImqQueue * queue = 0);

first distributed queue を設定します。

その他の関連クラス

ImqMessage

ImqQueue

ImqError クラス（C++）

この抽象クラスは，あるオブジェクトに関連づけられたエラーに関する情報を提供します。

インクルードファイル

このクラスを使用するときは，imqerr.hpp ファイルをインクルードしてください。

オブジェクト属性

- completion code

最新の完了コードです。初期値は 0 です。

- reason code

最新の理由コードです。初期値は 0 です。

コンストラクタ

- ImqError( );

デフォルトのコンストラクタです。

- ImqError(const ImqError & error);

コピーコンストラクタです。

メソッド

- void operator = (const ImqError & error);

インスタンスデータが error からコピーされ，既存のインスタンスデータと置き換えられます。

- void clearErrorCodes( );

completion code と reason code を両方とも 0 に設定します。

- MQLONG completionCode( ) const ;

completion code を返します。

- MQLONG reasonCode( ) const ;

reason code を返します。

メソッド（protected）

● ImqBoolean checkReadPointer(const void * pointer，const size_t length);

pointer と length の組み合わせが，読み取り専用アクセスに有効であるかどうかを調べます。正常に実行された場合は TRUE を返します。

● ImqBoolean checkWritePointer(const void * pointer，const size_t length);

pointer と length の組み合わせが，読み取りおよび書き込みアクセスに有効であるかどうかを調べます。正常に実行された場合は TRUE を返します。

● void setCompletionCode(const MQLONG code = 0);

completion code を設定します。

● void setReasonCode(const MQLONG code = 0);

reason code を設定します。

理由コード

• MQRC_BUFFER_ERROR

理由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。

ImqGetMessageOptions クラス（C++）

このクラスは，MQGMO 構造体をカプセル化します。

インクルードファイル

このクラスを使用するときは，imqgmo.hpp ファイルをインクルードしてください。

オブジェクト属性

- group status

メッセージのグループに関連するメッセージの状況です。初期値は MQGS_NOT_IN_GROUP です。

MQGMO 構造体の，GroupStatus フィールドに相当します。

- match options

着信メッセージを選択するためのオプションです。初期値は，MQMO_MATCH_MSG_ID または MQMO_MATCH_CORREL_ID です。

MQGMO 構造体の，MatchOptions フィールドに相当します。

- options

メッセージに適用できるオプションです。初期値は MQGMO_NO_WAIT です。

MQGMO 構造体の，Options フィールドに相当します。

- resolved queue name

解決されたキューの名前です。属性は読み取り専用です。名前は長さが 48 文字以下でなければならないので，その長さになるまでヌル文字が埋められます。初期値は空白です。

MQGMO 構造体の，ResolvedQName フィールドに相当します。

- segmentation

メッセージを分割する機能です。初期値は MSEG_INHIBITED です。

MQGMO 構造体の，Segmentation フィールドに相当します。

- segment status

メッセージの分割状況です。初期値は MQSS_NOT_A_SEGMENT です。

MQGMO 構造体の，SegmentStatus フィールドに相当します。

- sync-point participation

同期点制御を受けてメッセージが検索される場合は TRUE です。

- wait interval

適切なメッセージがまだ使用可能でない場合に，ImqQueue クラスの get メソッドが適切なメッセージの着信を待つ時間の長さです。初期値は 0 で，この場合は無期限に待機します。options に MQGMO_WAIT が組み込まれていない場合は，この属性は無視されます。

MQGMO 構造体の，WaitInterval フィールドに相当します。

コンストラクタ

- ImqGetMessageOptions( );

デフォルトのコンストラクタです。

- ImqGetMessageOptions(const ImqGetMessageOptions & gmo);

コピーコンストラクタです。

メソッド

- void operator = (const ImqGetMessageOptions & gmo);

インスタンスデータが gmo からコピーされ，既存のインスタンスデータと置き換えられます。

- MQCHAR groupStatus() const ;

group status を返します。

- void setGroupStatus(const MQCHAR status);

group status を設定します。

- MQLONG matchOptions() const ;

match options を返します。

- void setMatchOptions(const MQLONG options);

match options を設定します。

- MQLONG options( ) const ;

options を返します。

- void setOptions(const MQLONG options);

sync-point participation 値を組み込んで，options を設定します。

- ImqString resolvedQueueName( ) const ;

resolved queue name のコピーを返します。

- MQCHAR segmentation( ) const ;

segmentation を返します。

- void setSegmentation(const MQCHAR value);

segmentation を設定します。

- MQCHAR segmentStatus( ) const ;

segment status を返します。

- void setSegmentStatus(const MQCHAR status);

segment status を設定します。

- ImqBoolean syncPointParticipation( ) const ;

sync-point participation 値を返します。options に MQGMO_SYNCPOINT が組み込まれている場合は TRUE を返します。

- void setSyncPointParticipation(const ImqBoolean sync);

sync-point participation 値を設定します。options の MQGMO_SYNCPOINT フラグだけを変更します。

- MQLONG waitInterval( ) const ;

wait interval を返します。

- void setWaitInterval(const MQLONG interval);

wait interval を設定します。

オブジェクト属性（protected）

- PMQGMO opgmo

MQGMO 構造体のアドレスです。

- MQLONG olVersion

opgmo でアドレス指定された MQGMO 構造体のバージョン番号です。

その他の関連クラス

ImqString

ImqHeader クラス（C++）

この抽象クラスは，MQDLH 構造体，および MQRMH 構造体の共通機能をカプセル化します。

インクルードファイル

このクラスを使用するときは，imqhdr.hpp ファイルをインクルードしてください。

オブジェクト属性

● character set

元のコード化文字セット識別子です。初期値は MQCCSI_Q_MGR です。

MQDLH 構造体，MQRMH 構造体の，CodedCharSetId フィールドに相当します。

● encoding

元の符号化です。初期値は MQENC_NATIVE です。

MQDLH 構造体，MQRMH 構造体の，Encoding フィールドに相当します。

● format

元の形式です。初期値は MQFMT_NONE です。

MQDLH 構造体，MQRMH 構造体の，Format フィールドに相当します。

● header flags

初期値は次のとおりです。

- ImqDeadLetterHeader クラスのオブジェクト：0
- ImqIMSBridgeHeader クラスのオブジェクト：MQIIH_NONE
- ImqReferenceHeader クラスのオブジェクト：MQRMHF_LAST

MQRMH 構造体の，Flags フィールドに相当します。

コンストラクタ

● ImqHeader( );

デフォルトのコンストラクタです。

● ImqHeader(const ImqHeader & header);

コピーコンストラクタです。

メソッド

- void operator=(const ImqHeader & header);

インスタンスデータが header からコピーされ，既存のインスタンスデータと置き換えられます。

- virtual MQLONG characterSet( ) const;

character set を返します。

- virtual void setCharacterSet(const MQLONG ccsid = MQCCSI_Q_MGR);

character set を設定します。

- virtual MQLONG encoding( ) const ;

encoding を返します。

- virtual void setEncoding(const MQLONG encoding = MQENC_NATIVE);

encoding を設定します。

- virtual ImqString format( ) const ;

空白を含め，format のコピーを返します。

- virtual void setFormat(const char * name = 0);

format を設定し，空白で 8 文字までを埋めます。

- virtual MQLONG headerFlags() const ;

header flags を返します。

- virtual void setHeaderFlags(const MQLONG flags = 0);

header flags を設定します。

その他の関連クラス

ImqDeadLetterHeader

ImqItem

ImqMessage

ImqReferenceHeader

ImqString

ImqItem クラス（C++）

この抽象クラスは，一つのメッセージ内の一つの項目を表します。項目は，メッセージバッファ内で一つに連結されます。どの特殊化も，一つの構造体 ID で始まる特定の構造体と関連づけられます。

この抽象クラス内で多様メソッドを使用することによって，メッセージとの間で項目をコピーできます。ImqMessage クラスの readItem メソッドおよび writeItem メソッドでは，アプリケーションプログラムにとって無理のない別の方法でこれらの多様メソッドを呼び出せるようにします。

インクルードファイル

このクラスを使用するときは，imqitm.hpp ファイルをインクルードしてください。

オブジェクト属性

- structure id

構造体の始めにある，4 文字のストリングです。属性は読み取り専用です。

MQ* 構造体の，StrucId フィールドに相当します。

コンストラクタ

- ImqItem( );

デフォルトのコンストラクタです。

- ImqItem(const ImqItem & item);

コピーコンストラクタです。

メソッド

- static ImqBoolean structureIdIs(const char * structure-id-to-test , const ImqMessage & msg);

着信 msg 内の次の ImqItem の structure id が，structure-id-to-test と同じであれば，TRUE を返します。次の項目は，現在 ImqCache data pointer によってアドレス指定されているメッセージバッファの部分として識別されます。

- void operator = (const ImqItem & item);

インスタンスデータが item からコピーされ，既存のインスタンスデータと置き換えられます。

- virtual ImqBoolean copyOut(ImqMessage & msg) = 0 ;

このオブジェクトを次の項目として出力メッセージバッファに書き込み，既存の項目が

あればそれに付加します。書き込み操作が正常に実行されると，ImqCache data length は増やされます。正常に実行された場合は TRUE を返します。

特定のサブクラスを扱うときは，このメソッドを上書きしてください。

- virtual ImqBoolean pasteIn(ImqMessage & msg) = 0 ;

オブジェクトを着信メッセージバッファから読み取ります。

オブジェクトのサブクラスは，msg オブジェクトのメッセージバッファ内で次に検出された structure id と矛盾しないものでなければなりません。

msg オブジェクトの encoding は，MQENC_NATIVE でなければなりません。メッセージは，ImqMessage encoding を MQENC_NATIVE に設定し，ImqGetMessageOptions を使って検索してください。

読み取り操作が正常に実行されると，ImqCache data length は減少します。正常に実行された場合は TRUE を返します。

特定のサブクラスを扱うときは，このメソッドを上書きしてください。

その他の関連クラス

ImqCache

ImqError

ImqMessage

理由コード

- MQRC_ENCODING_ERROR
- MQRC_STRUC_ID_ERROR
- MQRC_INCONSISTENT_FORMAT
- MQRC_INSUFFICIENT_BUFFER
- MQRC_INSUFFICIENT_DATA

理由コードの詳細については，「付録 A　理由コード」を参照してください。

ImqMessage クラス（C++）

このクラスは，MQMD 構造体をカプセル化し，メッセージデータを構築および再構築します。

インクルードファイル

このクラスを使用するときは，imqmsg.hpp ファイルをインクルードしてください。

オブジェクト属性

- application id data

メッセージに関連づけられた ID 情報です。初期値は空白です。

MQMD 構造体の，ApplIdentityData フィールドに相当します。

- application origin data

メッセージに関連づけられた起点情報です。初期値は空白です。

MQMD 構造体の，ApplOriginData フィールドに相当します。

- backout count

メッセージが試験的に取り出され，次にロールバックされた回数です。属性は読み取り専用です。初期値は 0 です。

MQMD 構造体の，BackoutCount フィールドに相当します。

- character set

コード化文字セット ID です。初期値は MQCCSI_Q_MGR です。

MQMD 構造体の，CodedCharSetId フィールドに相当します。

- encoding

メッセージデータのマシン符号化です。初期値は MQENC_NATIVE です。

MQMD 構造体の，Encoding フィールドに相当します。

- expiry

TP1/Message Queue - Access が未検索メッセージを破棄する前に保存する期間を制御する時間，依存の数量です。初期値は MQEI_UNLIMITED です。

MQMD 構造体の，Expiry フィールドに相当します。

- format

バッファ内のデータのレイアウトを記述する形式（テンプレート）の名前です。9 文字以上の名前は，8 文字に切り捨てられます。名前は，8 文字の長さまで空白で埋められます。初期値は MQFMT_NONE です。

MQMD 構造体の，Format フィールドに相当します。

- message flags

分割制御情報です。初期値は MQMF_SEGMENTATION_INHIBITED です。

MQMD 構造体の，MsgFlags フィールドに相当します。

- message type

メッセージの広範囲なカテゴリ化です。初期値は MQMT_DATAGRAM です。

MQMD 構造体の，MsgType フィールドに相当します。

- offset

オフセット情報です。初期値は 0 です。

MQMD 構造体の，Offset フィールドに相当します。

- original length

分割されたメッセージの元の長さです。初期値は MQOL_UNDEFINED です。

MQMD 構造体の，OriginalLength フィールドに相当します。

- persistence

該当するメッセージが重大であり，持続記憶域を使用して，常にバックアップを取っておく必要があることを示します。このオプションを利用すると，パフォーマンスが低下します。初期値は MQPER_PERSISTENCE_AS_Q_DEF です。

MQMD 構造体の，Persistence フィールドに相当します。

- priority

伝送および送達の相対優先順位です。同じ優先順位のメッセージは，通常，提供されたのと同じ順序で送達されます。ただし，これが確実に実行されるようにするには，満たさなければならない基準が幾つかあります。初期値は MQPRI_PRIORITY_AS_Q_DEF です。

MQMD 構造体の，Priority フィールドに相当します。

- put application name

メッセージを書き込むアプリケーションの名前です。初期値はヌル文字です。

MQMD 構造体の，PutApplName フィールドに相当します。

- put application type

メッセージを書き込むアプリケーションのタイプです。初期値は MQAT_NO_CONTEXT です。

MQMD 構造体の，PutApplType フィールドに相当します。

- put date

メッセージが書き込まれた日付です。初期値は空白です。

MQMD 構造体の，PutDate フィールドに相当します。

- put time

メッセージが書き込まれた時刻です。初期値は空白です。

MQMD 構造体の，PutTime フィールドに相当します。

- reply-to queue manager name

応答メッセージが送られるキューマネジャの名前です。初期値はヌル文字です。

MQMD 構造体の，ReplyToQMgr フィールドに相当します。

- reply-to queue name

応答が送られるキューマネジャの名前です。初期値はヌル文字です。

MQMD 構造体の，ReplyToQ フィールドに相当します。

- report

メッセージと関連づけられているフィードバック情報です。初期値は MQRO_NONE です。

MQMD 構造体の，Report フィールドに相当します。

- sequence number

グループ内のメッセージを識別するシーケンス情報です。初期値は 1 です。

MQMD 構造体の，MsgSeqNumber フィールドに相当します。

- total message length

メッセージを最後に読み取ろうとしたときに有効だったバイト数です。属性は読み取り専用で，初期値は 0 です。

最後のメッセージに切り捨てが発生した場合，または切り捨てが発生したために読み取りが行われなかった場合，この値は ImqCache クラスの message length より大きくなり

ます。

- user id

メッセージと関連づけられているユーザ ID です。初期値はヌル文字です。

MQMD 構造体の，UserIdentifier フィールドに相当します。

コンストラクタ

- ImqMessage( );

デフォルトのコンストラクタです。

- ImqMessage(const ImqMessage & msg);

コピーコンストラクタです。詳細については，「メソッド」の operator の説明を参照してください。

メソッド

- void operator = (const ImqMessage & msg);

msg から MQMD 構造体，およびメッセージデータをコピーします。このオブジェクトのユーザによってバッファが提供されている場合は，コピーされるデータの量は使用できるバッファサイズまでに制限されます。そうでない場合は，システムは，必ず十分なサイズのバッファがコピーされたデータに使用できるようにします。

- ImqString applicationIdData( ) const ;

application id data のコピーを返します。

- void setApplicationIdData(const char * data = 0);

application id data を設定します。

- ImqString applicationOriginData( ) const ;

application origin data のコピーを返します。

- void setApplicationOriginData(const char * data = 0);

application origin data を設定します。

- MQLONG backoutCount( ) const ;

backout count を返します。

- MQLONG characterSet( ) const ;

character set を返します。

- void setCharacterSet(const MQLONG ccsid = MQCCSI_Q_MGR);

character set を設定します。

- MQLONG encoding( ) const ;

encoding を返します。

- void setEncoding(const MQLONG encoding = MQENC_NATIVE);

encoding を設定します。

- MQLONG expiry( ) const ;

expiry を返します。

- void setExpiry(const MQLONG expiry);

expiry を設定します。

- ImqString format( ) const ;

空白を含め，format のコピーを返します。

- ImqBoolean formatIs(const char * format-to-test) const ;

format が format-to-test と同じであれば，TRUE を返します。

- void setFormat(const char * name = 0);

format を設定し，空白で 8 文字までを埋めます。

- MQLONG messageFlags() const ;

message flags を返します。

- void setMessageFlags(const MQLONG flags);

message flags を設定します。

- MQLONG messageType( ) const ;

message type を返します。

- void setMessageType(const MQLONG type);

message type を設定します。

- MQLONG offset() const ;

offset を返します。

- void setOffset(const MQLONG offset);

offset を設定します。

- MQLONG originalLength() const ;

original length を返します。

- void setOriginalLength(const MQLONG length);

original length を設定します。

- MQLONG persistence( ) const ;

persistence を返します。

- void setPersistence(const MQLONG persistence);

persistence を設定します。

- MQLONG priority( ) const ;

priority を返します。

- void setPriority(const MQLONG priority);

priority を設定します。

- ImqString putApplicationName( ) const ;

put application name のコピーを返します。

- void setPutApplicationName(const char * name = 0);

put application name を設定します。

- MQLONG putApplicationType( ) const ;

put application type を返します。

- void setPutApplicationType(const MQLONG type = MQAT_NO_CONTEXT);

put application type を設定します。

- ImqString putDate( ) const ;

put date のコピーを返します。

- void setPutDate(const char * date = 0);

put date を設定します。

- ImqString putTime( ) const ;

put time のコピーを返します。

- void setPutTime(const char * time = 0);

put time を設定します。

- ImqBoolean readItem(ImqItem & item);

ImqItem pasteIn メソッドを使用して，メッセージバッファから item オブジェクトへ読み込みます。正常に実行された場合は TRUE を返します。

- ImqString replyToQueueManagerName( ) const ;

reply-to queue manager name のコピーを返します。

- void setReplyToQueueManagerName(const char * name = 0);

reply-to queue manager name を設定します。

- ImqString replyToQueueName( ) const ;

reply-to queue name のコピーを返します。

- void setReplyToQueueName(const char * name = 0);

reply-to queue name を設定します。

- MQLONG report( ) const ;

report を返します。

- void setReport(const MQLONG report);

report を設定します。

- MQLONG sequenceNumber() const ;

sequence number を返します。

- void setSequenceNumber(const MQLONG number);

sequence number を設定します。

- size_t totalMessageLength( ) const ;

total message length を返します。

- ImqString userId( ) const ;

user id のコピーを返します。

- void setUserId(const char * id = 0);

user id を設定します。

- ImqBoolean writeItem(ImqItem & item);

ImqItem copyOut メソッドを使用して，item オブジェクトからメッセージバッファへ書き込みます。書き込みは，挿入，置換，または付加の形態をとる場合がありますが，これは item オブジェクトのクラスによって決まります。正常に実行された場合は TRUE を返します。

オブジェクト属性（protected）

- MQMD omqmd

MQMD 構造体です。

その他の関連クラス

ImqCache

ImqItem

ImqMessageTracker

ImqString

理由コード

- MQRC_ENCODING_ERROR
- MQRC_STRUC_ID_ERROR
- MQRC_INSUFFICIENT_BUFFER
- MQRC_INSUFFICIENT_DATA

理由コードの詳細については，「付録 A　理由コード」を参照してください。

ImqMessageTracker クラス（C++）

この抽象クラスは，ImqMessage オブジェクトまたは ImqQueue オブジェクトの属性のうち，もう一方のオブジェクトと関連づけることのできる属性をカプセル化します。

インクルードファイル

このクラスを使用するときは，imqmtr.hpp ファイルをインクルードしてください。

オブジェクト属性

- accounting token

長さ MQ_ACCOUNTING_TOKEN_LENGTH の 2 進値（MQBYTE32）です。初期値は MQACT_NONE です。

MQMD 構造体，MQPMR 構造体の，AccountingToken フィールドに相当します。

- correlation id

メッセージを相互に関連づける目的でユーザが割り当てた，長さが MQ_CORREL_ID_LENGTH の 2 進値（MQBYTE24）です。初期値は MQCI_NONE です。

MQMD 構造体，MQPMR 構造体の，CorrelId フィールドに相当します。

- feedback

メッセージとともに送られるフィードバック情報です。初期値は MQFB_NONE です。

MQMD 構造体，MQPMR 構造体の，Feedback フィールドに相当します。

- group id

キュー内で固有の長さ MQ_GROUP_ID_LENGTH の 2 進値（MQBYTE24）です。初期値は MQGI_NONE です。

MQMD 構造体，MQPMR 構造体の，GroupId フィールドに相当します。

- message id

キュー内で固有の長さ MQ_MSG_ID_LENGTH の 2 進値（MQBYTE24）です。初期値は MQMI_NONE です。

MQMD 構造体，MQPMR 構造体の，MsgId フィールドに相当します。

コンストラクタ

- ImqMessageTracker( );

デフォルトのコンストラクタです。

- ImqMessageTracker(const ImqMessageTracker & tracker);

コピーコンストラクタです。詳細については，「メソッド」の operator の説明を参照してください。

メソッド

- void operator = (const ImqMessageTracker & tracker);

インスタンスデータが tracker からコピーされ，既存のインスタンスデータと置き換えられます。

- ImqBinary accountingToken( ) const ;

accounting token のコピーを返します。

- ImqBoolean setAccountingToken(const ImqBinary & token);

accounting token を設定します。token の data length は，0 または MQ_ACCOUNTING_TOKEN_ID_LENGTH のどちらかでなければなりません。正常に実行された場合は TRUE を返します。

- void setAccountingToken(const MQBYTE32 token = 0);

accounting token を設定します。token が 0 の場合は，MQACT_NONE を指定するのと同じです。token が 0 でない場合は，MQ_ACCOUNTING_TOKEN_LENGTH バイトの 2 進データをアドレス指定する必要があります。

MQACT_NONE などの事前定義値を使用する場合は，確実に信号機能が一致するようにキャストを作成する必要があります。

キャストは，例えば (MQBYTE *)MQACT_NONE のように作成します。

- ImqBinary correlationId( ) const ;

correlation id のコピーを返します。

- ImqBoolean setCorrelationId(const ImqBinary & token);

correlation id を設定します。token の data length は，0 または MQ_CORREL_ID_LENGTH のどちらかでなければなりません。正常に実行された場合は TRUE を返します。

● void setCorrelationId(const MQBYTE24 id = 0);

correlation id を設定します。id が 0 の場合は，MQCI_NONE を指定するのと同じです。id が 0 でない場合には，MQ_CORREL_ID_LENGTH バイトの 2 進データをアドレス指定する必要があります。MQCI_NONE などの事前定義値を使用する場合は，確実に信号機能が一致するようにキャストを作成する必要があります。

キャストは，例えば (MQBYTE *)MQCI_NONE のように作成します。

● MQLONG feedback( ) const ;

feedback を返します。

● void setFeedback(const MQLONG feedback);

feedback を設定します。

● ImqBinary groupId() const ;

group id のコピーを返します。

● ImqBoolean setGroupId(const ImqBinary & token);

group id を設定します。token の data length は，0 または MQ_GROUP_ID_LENGTH のどちらかでなければなりません。正常に実行された場合は TRUE を返します。

● void setGroupId(const MQBYTE24 id = 0);

group id を設定します。id が 0 の場合は，MQGI_NONE を指定するのと同じです。id が 0 でない場合は，MQ_GROUP_ID_LENGTH バイトの 2 進データをアドレス指定する必要があります。MQGI_NONE などの事前定義値を使用する場合は，確実に信号機能が一致するようにキャストを作成する必要があります。

キャストは，例えば (MQBYTE *)MQGI_NONE のように作成します。

● ImqBinary messageId( ) const ;

message id のコピーを返します。

● ImqBoolean setMessageId(const ImqBinary & token);

message id を設定します。token の data length は，0 または MQ_MSG_ID_LENGTH のどちらかでなければなりません。正常に実行された場合は TRUE を返します。

● void setMessageId(const MQBYTE24 id = 0);

message id を設定します。id が 0 の場合は，MQMI_NONE を指定するのと同じです。id が 0 でない場合には，MQ_MSG_ID_LENGTH バイトの 2 進データをアドレス指定する必要があります。MQMI_NONE などの事前定義値を使用する場合は，確実に信号機能が一致するようにキャストを作成する必要があります。

キャストは，例えば (MQBYTE*)MQMI_NONE のように作成します。

その他の関連クラス

ImqBinary

ImqError

ImqMessage

ImqQueue

理由コード

- MQRC_BINARY_DATA_LENGTH_ERROR

理由コードの詳細については，「付録 A　理由コード」を参照してください。

ImqObject クラス（C++）

このクラスは，キューマネジャおよびキューの属性をカプセル化します。

このクラスは抽象クラスです。このクラスのオブジェクトが破棄されると，このクラスは自動的にクローズされ，その ImqQueueManager 接続は切断されます。

インクルードファイル

このクラスを使用するときは，imqobj.hpp ファイルをインクルードしてください。

オブジェクト属性

- alternate user id

最大 MQ_USER_ID_LENGTH 文字です。初期値はヌル文字です。

MQOD 構造体の，AlternateUserId フィールドに相当します。

- close options

初期値は MQCO_NONE です。この属性は，暗黙の再オープン操作時には無視されます。この操作では，必ず値 MQCO_NONE が使用されるためです。

- connection reference

ローカルのキューマネジャへの必要な接続を提供する ImqQueueManager オブジェクトへの参照です。ImqQueueManager オブジェクトの場合，参照はオブジェクト自身です。初期値はヌル文字です。

注意

connection reference と，名前を指定されたキューのキューマネジャを識別する ImqQueue queue manager name とを混同しないようにしてください。

- description

キューマネジャ，キュー，またはプロセスの記述名（最大 64 文字）です。属性は読み取り専用です。

- name

キューマネジャ，キュー，または該当するプロセスの名前（最大 48 文字）です。初期値はヌル文字です。これは，結果として生じた動的キューの名前に対する open のあとのモデルキュー変更の名前です。空のキューマネジャ名の代わりに，必ず実際のキューマネジャ名が返されます。

MQOD 構造体，MQOR 構造体の，ObjectName フィールド，または ObjectQMgrName フィールドに相当します。

● next managed object

特定の順序でなく，このクラスの次のオブジェクトで，このオブジェクトと同じ connection reference を持つものです。初期値は 0 です。

● open options

初期値は MQOO_INQUIRE です。適切な値を設定する方法には，次の 2 通りがあります。

1. open options を設定しないで，open メソッドも使用しないでください。
   TP1/Message Queue - Access は，open options を自動的に調整し，必要に応じてオブジェクトの自動的なオープン，再オープン，およびクローズを実行します。この結果，不要な再オープン操作が実行される場合があります。これは，TP1/Message Queue - Access が openFor メソッドを使用するため，open options が次々に追加されるためです。
2. 結果的に MQI 命令が発生する任意のメソッドを使用する前に，open options を適切に設定してください。
   これによって，不要な再オープン操作が実行されなくなります。潜在的な再オープン問題が発生しやすい場合は，必ず，open options を明示的に設定してください。

open メソッドを使用する場合は，最初に open options が適切かどうか確認する必要があります。ただし，必ずしも open メソッドを使用する必要はありません。TP1/Message Queue - Access は，引き続き 1 の場合と同じ動作を示しますが，2 の状況では効率の低下にはなりません。

0 は有効な値ではないため，オブジェクトをオープンしてみる前に，適切な値を設定する必要があります。setOpenOptions(open options) のあとに open( ) を使用するか，openFor(open options) を使用して実行します。

● open status

オブジェクトがオープンである（TRUE）か，クローズされている（FALSE）かを示します。属性は読み取り専用です。初期値は FALSE です。

● previous managed object

特定の順序ではなく，このクラスの直前のオブジェクトで，このオブジェクトと同じ connection reference を持つものです。初期値は 0 です。

コンストラクタ

● ImqObject( );

デフォルトのコンストラクタです。

- ImqObject(const ImqObject & object);

コピーコンストラクタです。open status は FALSE となります。

メソッド

- void operator = (const ImqObject & object);

必要に応じてクローズを実行し，object からインスタンスデータをコピーします。open status は FALSE となります。

- ImqString alternateUserId( ) const ;

alternate user id のコピーを返します。

- ImqBoolean setAlternateUserId(const char * id);

alternate user id を設定します。alternate user id を設定できるのは，open status が FALSE である場合だけです。正常に実行された場合は TRUE を返します。

- ImqBoolean close( );

open status を FALSE に設定します。正常に実行された場合は TRUE を返します。

MQCLOSE 命令に相当します。

- MQLONG closeOptions( ) const ;

close options を返します。

- void setCloseOptions(const MQLONG options);

close options を設定します。

- ImqQueueManager * connectionReference( ) const ;

connection reference を返します。

- void setConnectionReference(ImqQueueManager & manager);

connection reference を設定します。

- void setConnectionReference(ImqQueueManager * manager = 0);

connection reference を設定します。

- virtual ImqBoolean description(ImqString & description) = 0 ;

description のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQCA_Q_MGR_DESC セレクタ，MQCA_Q_DESC セレクタ，および MQCA_PROCESS_DESC セレクタに相当します。

● ImqString description( );

考えられるエラーを指示しないで，description のコピーを返します。

MQINQ 命令の，MQCA_Q_MGR_DESC セレクタ，MQCA_Q_DESC セレクタ，および MQCA_PROCESS_DESC セレクタに相当します。

● ImqObject * nextManagedObject( ) const ;

next managed object を返します。

● virtual ImqBoolean name(ImqString & name);

name のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQCA_Q_MGR_NAME セレクタ，MQCA_Q_NAME セレクタ，および MQCA_PROCESS_NAME セレクタに相当します。

● ImqString name( );

考えられるエラーを指示しないで，name のコピーを返します。

MQINQ 命令の，MQCA_Q_MGR_NAME セレクタ，MQCA_Q_NAME セレクタ，および MQCA_PROCESS_NAME セレクタに相当します。

● ImqBoolean setName(const char * name = 0);

name を設定します。name を設定できるのは，open status が FALSE である場合だけです。また，ImqQueueManager の場合は，connection status が FALSE である場合です。正常に実行された場合は TRUE を返します。

● ImqBoolean open( );

open options と name を使用して必要に応じてオブジェクトをオープンすることによって，open status を TRUE に変更します。ImqQueueManager connection status を必ず TRUE にする必要がある場合には，connection reference 情報と ImqQueueManager connect メソッドを使用します。open status を返します。

MQOPEN 命令に相当します。

● ImqBoolean openFor(const MQLONG required-options = 0);

required-options が組み込まれた open options を指定した状態で，オブジェクトがオープンされるようにします。

● required-options が 0 である場合
入力は必須であり，どの入力オプションでも有効です。
- open options に MQOO_INPUT_AS_Q_DEF，MQOO_INPUT_SHARED，および MQOO_INPUT_EXCLUSIVE が含まれているときは，open options はすでに有効

であり，変更されません。
- open options に MQOO_INPUT_AS_Q_DEF，MQOO_INPUT_SHARED，および MQOO_INPUT_EXCLUSIVE のどれも含まれていないときは，open options に MQOO_INPUT_AS_Q_DEF が設定されます。

● required-options が 0 でない場合
必要なオプションが open options に追加されます。
- open options が MQOO_INPUT_AS_Q_DEF，MQOO_INPUT_SHARED，および MQOO_INPUT_EXCLUSIVE のどれかであるとき，それ以外の値はリセットされます。

open options のどれかが変更され，オブジェクトがすでにオープンになっている場合は，open options を調整するためにオブジェクトは一時的にクローズされ，再オープンされます。

正常に実行された場合は TRUE を返します。正常に実行されたということは，オブジェクトが適切なオプションでオープンになっていることを示します。

MQOPEN 命令に相当します。

● MQLONG openOptions( ) const ;

open options を返します。

● ImqBoolean setOpenOptions(const MQLONG options);

open options を設定します。open options を設定できるのは，open status が FALSE である場合だけです。正常に実行された場合は TRUE を返します。

● ImqBoolean openStatus( ) const ;

open status を返します。

● ImqObject * previousManagedObject( ) const ;

previous managed object を返します。

メソッド（protected）

● MQHCONN connectionHandle( ) const ;

connection reference と関連づけられた MQHCONN を返します。この値は，connection reference がない場合，または ImqQueueManager が接続されていない場合は 0 です。

● ImqBoolean inquire(const MQLONG int-attr, MQLONG & value);

整数値を返します。この索引は MQIA_* 値です。エラーがあった場合，値は MQIAV_UNDEFINED に設定されます。

● ImqBoolean inquire(const MQLONG char-attr，char * & buffer，const size_t length);

文字ストリングを返します。この索引は MQCA_* 値です。

注意

上記のメソッドは，どれも単一の属性値しか返しません。複数の値についてスナップショットが必要な場合，値は一つのインスタンスについて互いに一貫していますが，MQC クライアント機能が提供する C++ インタフェースではこの機能が用意されていないため，適切なパラメタを使って MQINQ 命令を使用する必要があります。

● virtual void openInformationDisperse( );

MQOPEN 命令の直後に MQOD 構造体の可変部分から情報を分散させます。

● virtual ImqBoolean openInformationPrepare( );

MQOPEN 命令の直前に MQOD 構造体の可変部分の情報を準備します。正常に実行された場合は TRUE を返します。

● ImqBoolean set(const MQLONG int-attr，const MQLONG value);

TP1/Message Queue - Access 整数属性を設定します。

● ImqBoolean set(const MQLONG char-attr，const char * buffer，const size_t required-length);

TP1/Message Queue - Access 文字属性を設定します。

● void setNextManagedObject(const ImqObject * object = 0);

next managed object を設定します。

● void setPreviousManagedObject(const ImqObject * object = 0);

previous managed object を設定します。

オブジェクト属性（protected）

● MQHOBJ ohobj

TP1/Message Queue - Access オブジェクトハンドルです。open status が TRUE である場合にだけ有効です。

● MQOD omqod

組み込み MQOD 構造体です。

その他の関連クラス

ImqBinary

ImqError

ImqQueue

ImqQueueManager

ImqString

理由コード

- MQRC_ATTRIBUTE_LOCKED
- MQRC_INCONSISTENT_OBJECT_STATE
- MQRC_NO_CONNECTION_REFERENCE
- MQRC_STORAGE_NOT_AVAILABLE
- （MQCLOSE からの理由コード）
- （MQCONN からの理由コード）
- （MQINQ からの理由コード）
- （MQOPEN からの理由コード）

理由コードの詳細については，「付録 A　理由コード」，またはマニュアル「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。

ImqProcess クラス（C++）

このクラスは，トリガモニタによってトリガできるアプリケーションプロセスの属性をカプセル化（オブジェクトタイプ MQOT_PROCESS）します。

インクルードファイル

このクラスを使用するときは，imqpro.hpp ファイルをインクルードしてください。

オブジェクト属性

- application id

アプリケーションプロセスの ID です。属性は読み取り専用です。

- application type

アプリケーションプロセスのタイプです。属性は読み取り専用です。

- environment data

プロセスの環境情報です。属性は読み取り専用です。

- user data

プロセスのユーザデータです。属性は読み取り専用です。

コンストラクタ

- ImqProcess( );

デフォルトのコンストラクタです。

- ImqProcess(const ImqProcess & process);

コピーコンストラクタです。ImqObject open status は FALSE です。

- ImqProcess(const char * name);

ImqObject name を設定します。

メソッド

- void operator = (const ImqProcess & process);

必要に応じてクローズを実行し，process からインスタンスデータをコピーします。ImqObject open status は FALSE です。

- ImqBoolean applicationId(ImqString & id);

application id のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQCA_APPL_ID セレクタに相当します。

- ImqString applicationId( );

考えられるエラーを指示しないで，application id を返します。

MQINQ 命令の，MQCA_APPL_ID セレクタに相当します。

- ImqBoolean applicationType(MQLONG & type);

application type のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_APPL_TYPE セレクタに相当します。

- MQLONG applicationType( );

考えられるエラーを指示しないで，application type を返します。

MQINQ 命令の，MQIA_APPL_TYPE セレクタに相当します。

- ImqBoolean environmentData(ImqString & data);

environment data のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQCA_ENV_DATA セレクタに相当します。

- ImqString environmentData( );

考えられるエラーを指示しないで，environment data を返します。

MQINQ 命令の，MQCA_ENV_DATA セレクタに相当します。

- ImqBoolean userData(ImqString & data);

user data のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQCA_USER_DATA セレクタに相当します。

- ImqString userData( );

考えられるエラーを指示しないで，user data を返します。

MQINQ 命令の，MQCA_USER_DATA セレクタに相当します。

その他の関連クラス

ImqObject

ImqPutMessageOptions クラス（C++）

このクラスは MQPMO 構造体をカプセル化します。

インクルードファイル

このクラスを使用するときは，imqpmo.hpp ファイルをインクルードしてください。

オブジェクト属性

- context reference

メッセージにコンテキストを提供する ImqQueue です。最初は，参照はありません。

MQPMO 構造体の，Context フィールドに相当します。

- options

書き込みメッセージオプションです。初期値は MQPMO_NONE です。

MQPMO 構造体の，Options フィールドに相当します。

- record fields

メッセージが書き込まれるときに PUT メッセージレコードの組み込みを制御するフラグです。初期値は MQPMRF_NONE です。

ImqMessageTracker 属性は，指定されたフィールドについては ImqQueue オブジェクトから取得されます。指定されないフィールドについては，ImqMessageTracker 属性は ImqMessage オブジェクトから取得されます。

MQPMO 構造体の，PutMsgRecFields フィールドに相当します。

- resolved queue manager name

書き込み中に判別された宛先キューマネジャの名前です。属性は読み取り専用です。初期値はヌル文字です。

MQPMO 構造体の，ResolvedQMgrName フィールドに相当します。

- resolved queue name

書き込み中に判別された宛先キューの名前です。属性は読み取り専用です。初期値はヌル文字です。

MQPMO 構造体の，ResolvedQName フィールドに相当します。

- sync-point participation

メッセージが同期点制御のときに書き込まれた場合は TRUE です。

コンストラクタ

- ImqPutMessageOptions( );

デフォルトのコンストラクタです。

- ImqPutMessageOptions(const ImqPutMessageOptions & pmo);

コピーコンストラクタです。

メソッド

- void operator = (const ImqPutMessageOptions & pmo);

インスタンスデータが pmo からコピーされ，既存のインスタンスデータと置き換えられます。

- ImqQueue * contextReference( ) const ;

context reference を返します。

- void setContextReference(const ImqQueue & queue);

context reference を設定します。

- void setContextReference(const ImqQueue * queue = 0);

context reference を設定します。

- MQLONG options( ) const ;

options を返します。

- void setOptions(const MQLONG options);

sync-point participation 値を組み込んで，options を設定します。

- MQLONG recordFields() const ;

record fields を返します。

- void setRecordFields(const MQLONG fields);

record fields を設定します。

- ImqString resolvedQueueManagerName( ) const ;

resolved queue manager name のコピーを返します。

- ImqString resolvedQueueName( ) const ;

resolved queue name のコピーを返します。

● ImqBoolean syncPointParticipation( ) const ;

sync-point participation 値を返します。options に MQPMO_SYNCPOINT が組み込まれている場合は TRUE を返します。

● void setSyncPointParticipation(const ImqBoolean sync);

sync-point participation 値を設定します。options の MQPMO_SYNCPOINT フラグだけを変更します。

オブジェクト属性（protected）

● MQPMO omqpmo

MQPMO 構造体です。

その他の関連クラス

ImqError

ImqMessage

ImqQueue

ImqString

理由コード

• MQRC_STORAGE_NOT_AVAILABLE

理由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。

ImqQueue クラス（C++）

このクラスは，メッセージキューをカプセル化（オブジェクトタイプ MQOT_Q）します。

インクルードファイル

このクラスを使用するときは，imqque.hpp ファイルをインクルードしてください。

オブジェクト属性

- base queue name

別名の元となるキューの名前です。属性は読み取り専用です。

- cluster name

クラスタの名前です。属性は読み取り専用です。

- creation date

キュー作成データです。属性は読み取り専用です。

- creation time

キュー作成時刻です。属性は読み取り専用です。

- current depth

キュー上にあるメッセージの数です。属性は読み取り専用です。

- default bind

デフォルトのバインドです。属性は読み取り専用です。

- default input open option

デフォルトの入力用オープンオプションです。属性は読み取り専用です。

- default persistence

デフォルトのメッセージ持続性です。属性は読み取り専用です。

- default priority

デフォルトのメッセージ優先順位です。属性は読み取り専用です。

- definition type

キューの定義タイプです。属性は読み取り専用です。

- distribution list reference

このキューを含む，複数のキューにメッセージを配布するのに使用できる ImqDistributionList への任意選択の参照です。初期値はヌル文字です。

ImqQueue オブジェクトがオープンされると，任意でオープンしていて，ImqQueue オブジェクトが参照する ImqDistributionList オブジェクトは，自動的にクローズされます。

- distribution lists

配布リストをサポートするための転送キューの機能です。属性は読み取り専用です。

- dynamic queue name

動的キュー名です。初期値は，すべてのパーソナルコンピュータ，および UNIX プラットフォームについて "AMQ.*" です。

MQOD 構造体の，DynamicQName フィールドに相当します。

- harden get backout

ロールバックカウントを固定するかどうかを判別します。属性は読み取り専用です。

- inhibit get

読み取り操作が許されているかどうかを判別します。初期値はキューの定義によって異なります。別名またはローカルキューについてだけ有効です。

- inhibit put

書き込み操作が許されているかどうかを判別します。初期値はキューの定義によって異なります。

- initiation queue name

開始キューの名前です。属性は読み取り専用です。

- maximum depth

キューで許されるメッセージの最大数です。属性は読み取り専用です。

- maximum message length

キュー上のあらゆるメッセージの最大長です。関連づけられているキューマネジャによって管理されるどのキューの最大長の値より小さいことがあります。属性は読み取り専用です。

- message delivery sequence

メッセージ優先順位が関係あるかどうかを判別します。属性は読み取り専用です。

- next distributed queue

ImqQueue クラスの，ImqDistributionList クラスに設定されている順序ではなく，このクラスの次のオブジェクトで，このオブジェクトと同じ distribution list reference を持つキューです。初期値は 0 です。

- open input count

入力できるようにオープンになっている ImqQueue オブジェクトの数です。属性は読み取り専用です。

- open output count

出力できるようにオープンになっている ImqQueue オブジェクトの数です。属性は読み取り専用です。

- previous distributed queue

ImqQueue クラスの，ImqDistributionList クラスに設定されている順序ではなく，このクラスの直前のオブジェクトで，このオブジェクトと同じ distribution list reference を持つキューです。初期値は 0 です。

- process name

プロセス定義の名前です。属性は読み取り専用です。

- queue manager name

キューが実際に入っているキューマネジャの名前です。キューが実際に入っているのは，多くの場合リモートキューマネジャです。ここに指定するキューマネジャを，接続を提供するキューマネジャを参照する ImqObject クラスの connection reference メソッドと混同しないでください。初期値はヌル文字です。

MQOD 構造体，MQOR 構造体の，ObjectQMgrName フィールドに相当します。

- queue type

キュータイプです。属性は読み取り専用です。

- remote queue manager name

リモートキューマネジャの名前です。属性は読み取り専用です。

- remote queue name

リモートキューマネジャで認識されているとおりのリモートキューの名前です。属性は読み取り専用です。

- resolved queue manager name

解決済みのキューマネジャの名前です。属性は読み取り専用です。

MQOD 構造体の，ResolvedQMgrName フィールドに相当します。

- resolved queue name

解決済みのキューの名前です。属性は読み取り専用です。

MQOD 構造体の，ResolvedQName フィールドに相当します。

- scope

キュー定義の有効範囲です。属性は読み取り専用です。

- shareability

キューを共用できるかどうかを判別します。属性は読み取り専用です。

- transmission queue name

転送キューの名前です。属性は読み取り専用です。

- trigger control

トリガ制御です。初期値は，キュー定義によって異なります。ローカルキューについてだけ有効です。

- trigger data

トリガデータです。初期値は，キュー定義によって異なります。ローカルキューについてだけ有効です。

- trigger depth

トリガサイズです。初期値はキュー定義によって異なります。ローカルキューについてだけ有効です。

- trigger message priority

トリガのしきい値メッセージ優先順位です。初期値はキュー定義によって異なります。ローカルキューについてだけ有効です。

- trigger type

トリガタイプです。初期値はキュー定義によって異なります。ローカルキューについてだけ有効です。

- usage

使用方法です。属性は読み取り専用です。

コンストラクタ

- ImqQueue( );

デフォルトのコンストラクタです。

- ImqQueue(const ImqQueue & queue);

コピーコンストラクタです。ImqObject open status は FALSE です。

- ImqQueue(const char * name);

ImqObject name を設定します。

メソッド

- void operator = (const ImqQueue & queue);

必要に応じてクローズを実行し，queue からインスタンスデータをコピーします。
ImqObject open status は FALSE です。

- ImqBoolean baseQueueName(ImqString & name);

base queue name のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQCA_BASE_Q_NAME セレクタに相当します。

- ImqString baseQueueName( );

考えられるエラーを指示しないで，base queue name を返します。

MQINQ 命令の，MQCA_BASE_Q_NAME セレクタに相当します。

- ImqBoolean clusterName(ImqString & name);

cluster name のコピーを提供します。正常に終了した場合は TRUE を返します。

MQINQ 命令の，MQCA_CLUSTER_NAME セレクタに相当します。

- ImqString clusterName();

考えられるエラーを指示しないで，cluster name を返します。

MQINQ 命令の，MQCA_CLUSTER_NAME セレクタに相当します。

- ImqBoolean creationDate(ImqString & date);

creation date のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQCA_CREATION_DATE セレクタに相当します。

- ImqString creationDate( );

考えられるエラーを指示しないで，creation date を返します。

MQINQ 命令の，MQCA_CREATION_DATE セレクタに相当します。

- ImqBoolean creationTime(ImqString & time);

creation time のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQCA_CREATION_TIME セレクタに相当します。

- ImqString creationTime( );

考えられるエラーを指示しないで，creation time を返します。

MQINQ 命令の，MQCA_CREATION_TIME セレクタに相当します。

- ImqBoolean currentDepth(MQLONG & depth);

current depth のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_CURRENT_Q_DEPTH セレクタに相当します。

- MQLONG currentDepth( );

考えられるエラーを指示しないで，current depth を返します。

MQINQ 命令の，MQIA_CURRENT_Q_DEPTH セレクタに相当します。

- ImqBoolean defaultInputOpenOption(MQLONG & option);

default input open option のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_DEF_INPUT_OPEN_OPTION セレクタに相当します。

- MQLONG defaultInputOpenOption( );

考えられるエラーを指示しないで，default input open option を返します。

MQINQ 命令の，MQIA_DEF_INPUT_OPEN_OPTION セレクタに相当します。

- ImqBoolean defaultPersistence(MQLONG & persistence);

default persistence のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_DEF_PERSISTENCE セレクタに相当します。

- MQLONG defaultPersistence( );

考えられるエラーを指示しないで，default persistence を返します。

MQINQ 命令の，MQIA_DEF_PERSISTENCE セレクタに相当します。

- ImqBoolean defaultPriority(MQLONG & priority);

default priority のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_DEF_PRIORITY セレクタに相当します。

- MQLONG defaultPriority( );

考えられるエラーを指示しないで，default priority を返します。

MQINQ 命令の，MQIA_DEF_PRIORITY セレクタに相当します。

- ImqBoolean defaultBind(MQLONG & bind);

default bind のコピーを提供します。正常に終了した場合は TRUE を返します。

MQINQ 命令の，MQIA_DEF_BIND セレクタに相当します。

- MQLONG defaultBind();

考えられるエラーを指示しないで，default bind を返します。

MQINQ 命令の，MQIA_DEF_BIND セレクタに相当します。

- ImqBoolean definitionType(MQLONG & type);

definition type のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_DEFINITION_TYPE セレクタに相当します。

- MQLONG definitionType( );

考えられるエラーを指示しないで，definition type を返します。

MQINQ 命令の，MQIA_DEFINITION_TYPE セレクタに相当します。

- ImqDistributionList * distributionListReference() const ;

distribution list reference を返します。

- void setDistributionListReference(ImqDistributionList & list);

distribution list reference を設定します。

- void setDistributionListReference(ImqDistributionList * list = 0);

distribution list reference を設定します。

- ImqBoolean distributionLists(MQLONG & support);

distribution lists 値のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_DIST_LISTS セレクタに相当します。

- MQLONG distributionLists();

考えられるエラーを指示しないで，distribution lists 値を返します。

MQINQ 命令の，MQIA_DIST_LISTS セレクタに相当します。

- ImqBoolean setDistributionLists(const MQLONG support);

distribution lists 値を設定します。正常に実行された場合は TRUE を返します。

MQSET 命令の，MQIA_DIST_LISTS セレクタに相当します。

- ImqString dynamicQueueName( ) const ;

dynamic queue name のコピーを返します。

- ImqBoolean setDynamicQueueName(const char * name);

dynamic queue name を設定します。dynamic queue name を設定できるのは，ImqObject open status が FALSE である場合だけです。正常に実行された場合は TRUE を返します。

- ImqBoolean get(ImqMessage & msg);

デフォルトの読み取りメッセージオプションを使用して，キューからメッセージを検索します。ImqObject open options に MQOO_INPUT_* 値のどれかが必ず含まれるようにする必要がある場合は，ImqObject openFor メソッドを使用します。検索の前に，msg オブジェクトに対して clearMessage メソッドが呼び出されます。正常に実行された場合は TRUE を返します。

MQGET 命令に相当します。

注意

MQRC_TRUNCATED_MSG_FAILED という ImqObject reason code は，警告ではなく，障害として扱われます。切り捨てられたメッセージが受け入れられる場合，ImqCache message length は切り捨てられた長さを反映します。

- ImqBoolean get(ImqMessage & msg，ImqGetMessageOptions & options);

指定された読み取りメッセージオプションを使用して，キューからメッセージを検索します。ImqGetMessageOptions options に応じて，ImqObject open options に，MQOO_INPUT_* 値のどれか，または MQOO_BROWSE 値のどちらかが必ず含まれるようにする必要がある場合は ImqObject openFor メソッドを使用します。検索の前に，msg オブジェクトに対して clearMessage メソッドが呼び出されます。正常に実行された場合は TRUE を返します。

MQGET 命令に相当します。

注意

MQRC_TRUNCATED_MSG_FAILED という ImqObject reason code は，警告ではなく，障害として扱われます。切り捨てられたメッセージが受け入れられる場合，ImqCache message length は切り捨てられた長さを反映します。

● ImqBoolean get(ImqMessage & msg, ImqGetMessageOptions & options, const size_t buffer-size);

buffer size で指定を変更できること以外は，ImqBoolean get(ImqMessage & msg); メソッド，および ImqBoolean get(ImqMessage & msg, ImqGetMessageOptions & options); メソッドと同じです。msg オブジェクトが ImqCache の automatic buffer を使用している場合は，メッセージを検索する前に，msg オブジェクトに対して resizeBuffer メソッドが呼び出されます。このバッファは検索されたメッセージに合わせて大きくなりません。

MQGET 命令に相当します。

● ImqBoolean get(ImqMessage & msg, const size_t buffer-size);

デフォルトの読み取りメッセージオプションが使用されること以外は，ImqBoolean get(ImqMessage & msg, ImqGetMessageOptions & options, const size_t buffer-size); メソッドと同じです。

MQGET 命令に相当します。

● ImqBoolean hardenGetBackout(MQLONG & harden);

harden get backout 値のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_HARDEN_GET_BACKOUT セレクタに相当します。

● MQLONG hardenGetBackout( );

考えられるエラーを指示しないで，harden get backout 値を返します。

MQINQ 命令の，MQIA_HARDEN_GET_BACKOUT セレクタに相当します。

● ImqBoolean inhibitGet(MQLONG & inhibit);

inhibit get 値のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_INHIBIT_GET セレクタに相当します。

● MQLONG inhibitGet( );

考えられるエラーを指示しないで，inhibit get 値を返します。

MQINQ 命令の，MQIA_INHIBIT_GET セレクタに相当します。

● ImqBoolean setInhibitGet(const MQLONG inhibit);

inhibit get 値を設定します。正常に実行された場合は TRUE を返します。

MQSET 命令の，MQIA_INHIBIT_GET セレクタに相当します。

- ImqBoolean inhibitPut(MQLONG & inhibit);

inhibit put 値のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_INHIBIT_PUT セレクタに相当します。

- MQLONG inhibitPut( );

考えられるエラーを指示しないで，inhibit put 値を返します。

MQINQ 命令の，MQIA_INHIBIT_PUT セレクタに相当します。

- ImqBoolean setInhibitPut(const MQLONG inhibit);

inhibit put 値を設定します。正常に実行された場合は TRUE を返します。

MQSET 命令の，MQIA_INHIBIT_PUT セレクタに相当します。

- ImqBoolean initiationQueueName(ImqString & name);

initiation queue name のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQCA_INITIATION_Q_NAME セレクタに相当します。

- ImqString initiationQueueName( );

考えられるエラーを指示しないで，initiation queue name を返します。

MQINQ 命令の，MQCA_INITIATION_Q_NAME セレクタに相当します。

- ImqBoolean maximumDepth(MQLONG & depth);

maximum depth のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_MAX_Q_DEPTH セレクタに相当します。

- MQLONG maximumDepth( );

考えられるエラーを指示しないで，maximum depth を返します。

MQINQ 命令の，MQIA_MAX_Q_DEPTH セレクタに相当します。

- ImqBoolean maximumMessageLength(MQLONG & length);

maximum message length のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_MAX_MSG_LENGTH セレクタに相当します。

- MQLONG maximumMessageLength( );

考えられるエラーを指示しないで，maximum message length を返します。

MQINQ 命令の，MQIA_MAX_MSG_LENGTH セレクタに相当します。

● ImqBoolean messageDeliverySequence(MQLONG & sequence);

message delivery sequence のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_MSG_DELIVERY_SEQUENCE セレクタに相当します。

● MQLONG messageDeliverySequence( );

考えられるエラーを指示しないで，message delivery sequence 値を返します。

MQINQ 命令の，MQIA_MSG_DELIVERY_SEQUENCE セレクタに相当します。

● ImqQueue * nextDistributedQueue() const ;

next distributed queue を返します。

● ImqBoolean openInputCount(MQLONG & count);

open input count のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_OPEN_INPUT_COUNT セレクタに相当します。

● MQLONG openInputCount( );

考えられるエラーを指示しないで，open input count を返します。

MQINQ 命令の，MQIA_OPEN_INPUT_COUNT セレクタに相当します。

● ImqBoolean openOutputCount(MQLONG & count);

open output count のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_OPEN_OUTPUT_COUNT セレクタに相当します。

● MQLONG openOutputCount( );

考えられるエラーを指示しないで，open output count を返します。

MQINQ 命令の，MQIA_OPEN_OUTPUT_COUNT セレクタに相当します。

● ImqQueue * previousDistributedQueue() const ;

previous distributed queue を返します。

● ImqBoolean processName(ImqString & name);

process name のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQCA_PROCESS_NAME セレクタに相当します。

● ImqString processName( );

考えられるエラーを指示しないで，process name を返します。

MQINQ 命令の，MQCA_PROCESS_NAME セレクタに相当します。

● ImqBoolean put(ImqMessage & msg);

デフォルトの書き込みメッセージオプションを使用して，キューにメッセージを配置します。ImqObject open options に MQOO_OUTPUT 値が必ず含まれるようにする必要がある場合は，ImqObject openFor メソッドを使用します。正常に実行された場合は TRUE を返します。

MQPUT 命令に相当します。

● ImqBoolean put(ImqMessage & msg，ImqPutMessageOptions & pmo);

指定された pmo を使用して，キューにメッセージを配置します。次の条件が満たされた場合に，対応する MQOO_*_CONTEXT 値が必ず含まれるようにする必要があれば，ImqObject openFor メソッドを使用します。

- ImqObject open options に MQOO_OUTPUT が含まれている場合
- pmo options に MQPMO_PASS_IDENTITY_CONTEXT，MQPMO_PASS_ALL_CONTEXT，MQPMO_SET_IDENTITY_CONTEXT，または MQPMO_SET_ALL_CONTEXT のどれかが含まれている場合

正常に実行された場合は TRUE を返します。

MQPUT 命令に相当します。

注意

pmo に context reference が一つ含まれている場合，コンテキストを提供する必要があれば，参照されるオブジェクトはオープンされます。

● ImqString queueManagerName( ) const ;

queue manager name を返します。

● ImqBoolean setQueueManagerName(const char * name);

queue manager name を設定します。queue manager name を設定できるのは，ImqObject open status が FALSE である場合だけです。正常に実行された場合は TRUE を返します。

● ImqBoolean queueType(MQLONG & type);

queue type 値のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_Q_TYPE セレクタに相当します。

● MQLONG queueType( );

考えられるエラーを指示しないで，queue type を返します。

MQINQ 命令の，MQIA_Q_TYPE セレクタに相当します。

● ImqBoolean remoteQueueManagerName(ImqString & name);

remote queue manager name のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQCA_REMOTE_Q_MGR_NAME セレクタに相当します。

● ImqString remoteQueueManagerName( );

考えられるエラーを指示しないで，remote queue manager name を返します。

MQINQ 命令の，MQCA_REMOTE_Q_MGR_NAME セレクタに相当します。

● ImqBoolean remoteQueueName(ImqString & name);

remote queue name のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQCA_REMOTE_Q_NAME セレクタに相当します。

● ImqString remoteQueueName( );

考えられるエラーを指示しないで，remote queue name を返します。

MQINQ 命令の，MQCA_REMOTE_Q_NAME セレクタに相当します。

● ImqBoolean resolvedQueueManagerName(ImqString & name);

resolved queue manager name のコピーを提供します。正常に終了した場合は TRUE を返します。

● ImqString resolvedQueueManagerName() const ;

考えられるエラーを指示しないで，resolved queue name を返します。

● ImqBoolean resolvedQueueName(ImqString & name);

resolved queue name のコピーを提供します。正常に終了した場合は TRUE を返します。

● ImqString resolvedQueueName() const ;

考えられるエラーを指示しないで，resolved queue name を返します。

● ImqBoolean scope(MQLONG & scope);

scope のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_SCOPE セレクタに相当します。

- MQLONG scope( );

考えられるエラーを指示しないで，scope を返します。

MQINQ 命令の，MQIA_SCOPE セレクタに相当します。

- ImqBoolean shareability(MQLONG & shareability);

shareability 値のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_SHAREABILITY セレクタに相当します。

- MQLONG shareability( );

考えられるエラーを指示しないで，shareability 値を返します。

MQINQ 命令の，MQIA_SHAREABILITY セレクタに相当します。

- ImqBoolean transmissionQueueName(ImqString & name);

transmission queue name のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQCA_XMIT_Q_NAME セレクタに相当します。

- ImqString transmissionQueueName( );

考えられるエラーを指示しないで，transmission queue name を返します。

MQINQ 命令の，MQCA_XMIT_Q_NAME セレクタに相当します。

- ImqBoolean triggerControl(MQLONG & control);

trigger control 値のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_TRIGGER_CONTROL セレクタに相当します。

- MQLONG triggerControl( );

考えられるエラーを指示しないで，trigger control 値を返します。

MQINQ 命令の，MQIA_TRIGGER_CONTROL セレクタに相当します。

- ImqBoolean setTriggerControl(const MQLONG control);

trigger control 値を設定します。正常に実行された場合は TRUE を返します。

MQSET 命令の，MQIA_TRIGGER_CONTROL セレクタに相当します。

- ImqBoolean triggerData(ImqString & data);

trigger data のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQCA_TRIGGER_DATA セレクタに相当します。

- ImqString triggerData( );

考えられるエラーを指示しないで，trigger data のコピーを返します。

MQINQ 命令の，MQCA_TRIGGER_DATA セレクタに相当します。

- ImqBoolean setTriggerData(const char * data);

trigger data を設定します。正常に実行された場合は TRUE を返します。

MQSET 命令の，MQCA_TRIGGER_DATA セレクタに相当します。

- ImqBoolean triggerDepth(MQLONG & depth);

trigger depth のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_TRIGGER_DEPTH セレクタに相当します。

- MQLONG triggerDepth( );

考えられるエラーを指示しないで，trigger depth を返します。

MQINQ 命令の，MQIA_TRIGGER_DEPTH セレクタに相当します。

- ImqBoolean setTriggerDepth(const MQLONG depth);

trigger depth を設定します。正常に実行された場合は TRUE を返します。

MQSET 命令の，MQIA_TRIGGER_DEPTH セレクタに相当します。

- ImqBoolean triggerMessagePriority(MQLONG & priority);

trigger message priority のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_TRIGGER_MSG_PRIORITY セレクタに相当します。

- MQLONG triggerMessagePriority( );

考えられるエラーを指示しないで，trigger message priority を返します。

MQINQ 命令の，MQIA_TRIGGER_MSG_PRIORITY セレクタに相当します。

- ImqBoolean setTriggerMessagePriority(const MQLONG priority);

trigger message priority を設定します。正常に実行された場合は TRUE を返します。

MQSET 命令の，MQIA_TRIGGER_MSG_PRIORITY セレクタに相当します。

- ImqBoolean triggerType(MQLONG & type);

trigger type のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_TRIGGER_TYPE セレクタに相当します。

- MQLONG triggerType( );

考えられるエラーを指示しないで，trigger type を返します。

MQINQ 命令の，MQIA_TRIGGER_TYPE セレクタに相当します。

- ImqBoolean setTriggerType(const MQLONG type);

trigger type を設定します。正常に実行された場合は TRUE を返します。

MQSET 命令の，MQIA_TRIGGER_TYPE セレクタに相当します。

- ImqBoolean usage(MQLONG & usage);

usage 値のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_USAGE セレクタに相当します。

- MQLONG usage( );

考えられるエラーを指示しないで，usage 値を返します。

MQINQ 命令の，MQIA_USAGE セレクタに相当します。

メソッド（protected）

- void setNextDistributedQueue(ImqQueue * queue = 0);

next distributed queue を設定します。

- void setPreviousDistributedQueue(ImqQueue * queue = 0);

previous distributed queue を設定します。

その他の関連クラス

ImqCache

ImqDistributionList

ImqGetMessageOptions

ImqMessage

ImqMessageTracker

ImqObject

ImqPutMessageOptions

ImqQueueManager

ImqString

理由コード

- MQRC_CONTEXT_OBJECT_NOT_VALID
- MQRC_CONTEXT_OPEN_ERROR
- MQRC_CURSOR_NOT_VALID
- MQRC_NO_BUFFER
- MQRC_REOPEN_EXCL_INPUT_ERROR
- MQRC_REOPEN_INQUIRE_ERROR
- MQRC_REOPEN_SAVED_CONTEXT_ERR
- MQRC_REOPEN_TEMPORARY_Q_ERROR
- （MQGET からの理由コード）
- （MQPUT からの理由コード）

理由コードの詳細については，「付録 A　理由コード」，またはマニュアル「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。

ImqQueueManager クラス（C++）

このクラスは，キューマネジャをカプセル化（オブジェクト MQOT_Q_MGR）します。

インクルードファイル

このクラスを使用するときは，imqmgr.hpp ファイルをインクルードしてください。

オブジェクト属性

- authority event

許可事象を制御します。属性は読み取り専用です。

- begin options

begin メソッドに適用されるオプションです。初期値は MQBO_NONE です。

MQBO 構造体の，Options フィールドに相当します。

- character set

コード化文字セット識別子です。属性は読み取り専用です。

- command input queue name

システムコマンド入力キュー名です。属性は読み取り専用です。

- command level

キューマネジャによってサポートされているコマンドレベルです。属性は読み取り専用です。

- connect options

connect メソッドに適用されるオプションです。初期値は MQCNO_NONE です。

- connection status

TRUE か FALSE かをメソッドに返します。キューマネジャに接続されている場合は，TRUE です。属性は読み取り専用です。

- dead-letter queue name

送達不能キューの名前です。属性は読み取り専用です。

- default transmission queue name

デフォルトの転送キュー名です。属性は読み取り専用です。

- distribution lists

配布リストをサポートするためのキューマネジャの機能です。属性は読み取り専用です。

- first managed object

ImqObject クラスの connection reference メソッドにアドレス指定される複数のオブジェクトのうち，先頭になるオブジェクトです。初期値は 0 です。

- inhibit event

禁止事象を制御します。属性は読み取り専用です。

- local event

ローカル事象を制御します。属性は読み取り専用です。

- maximum handles

ハンドルの最大数です。属性は読み取り専用です。

- maximum message length

キューマネジャによって管理される任意のキュー上にある任意のメッセージの最大長です。属性は読み取り専用です。

- maximum priority

最高のメッセージ優先順位です。属性は読み取り専用です。

- maximum uncommitted messages

一つのトランザクション内のコミットされていないメッセージの最大数です。属性は読み取り専用です。

- performance event

パフォーマンス事象を制御します。属性は読み取り専用です。

- platform

キューマネジャが収容されているプラットフォームです。属性は読み取り専用です。

- remote event

リモート事象を制御します。属性は読み取り専用です。

- repository name

リポジトリの名前です。属性は読み取り専用です。

- start-stop event

調歩式事象を制御します。属性は読み取り専用です。

- sync-point availability

同期点参加の可用性です。属性は読み取り専用です。

- trigger interval

トリガ間隔です。属性は読み取り専用です。

コンストラクタ

- ImqQueueManager( );

デフォルトのコンストラクタです。

- ImqQueueManager(const ImqQueueManager & manager);

コピーコンストラクタです。connection status は FALSE です。

- ImqQueueManager(const char * name);

ImqObject name を name に設定します。

デストラクタ

ImqQueueManager オブジェクトは，破棄されると自動的に切断されます。

メソッド

- void operator = (const ImqQueueManager & mgr);

必要に応じて切断し，次に，mgr からインスタンスデータをコピーします。connection status は FALSE です。

- ImqBoolean backout( );

コミットされていない変更内容をロールバックします。正常に実行された場合は TRUE を返します。

MQBACK 命令に相当します。

- ImqBoolean begin( );

一つのトランザクションを始めます。begin options は，このメソッドの動作に影響します。正常に実行された場合は TRUE を返します。

MQBEGIN 命令に相当します。

- MQLONG beginOptions( ) const ;

begin options を返します。

● void setBeginOptions(const MQLONG options = MQBO_NONE);

begin options を設定します。

● ImqBoolean characterSet(MQLONG & ccsid);

character set のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_CODED_CHAR_SET_ID セレクタに相当します。

● MQLONG characterSet( );

考えられるエラーを指示しないで，character set のコピーを返します。

MQINQ 命令の，MQIA_CODED_CHAR_SET_ID セレクタに相当します。

● ImqBoolean commit( );

コミットされていない変更内容をコミットします。正常に実行された場合は TRUE を返します。

MQCMIT 命令に相当します。

● ImqBoolean connect( );

与えられた ImqObject name を持つキューマネジャへ接続します。デフォルトはローカルキューマネジャです。特定のキューマネジャに接続したい場合は，接続前に ImqObject setName メソッドを使用してください。connect options は，このメソッドの動作に影響します。connection status を TRUE に設定します。正常に実行された場合は TRUE を返します。

MQCONN 命令に相当します。

注意

同一のキューマネジャに複数の ImqQueueManager オブジェクトを接続できますが，各接続は，別個のスレッドから実行する必要があります。

● ImqBoolean connectionStatus( ) const ;

connection status を返します。

● ImqBoolean deadLetterQueueName(ImqString & name);

dead-letter queue name のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQCA_DEAD_LETTER_Q_NAME セレクタに相当します。

● ImqString deadLetterQueueName( );

考えられるエラーを指示しないで，dead-letter queue name のコピーを返します。

MQINQ 命令の，MQCA_DEAD_LETTER_Q_NAME セレクタに相当します。

● ImqBoolean defaultTransmissionQueueName(ImqString & name);

default transmission queue name のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQCA_DEF_XMIT_Q_NAME セレクタに相当します。

● ImqString defaultTransmissionQueueName( );

考えられるエラーを指示しないで，default transmission queue name を返します。

MQINQ 命令の，MQCA_DEF_XMIT_Q_NAME セレクタに相当します。

● ImqBoolean disconnect( );

キューマネジャから切断し，connection status を FALSE に設定します。このオブジェクトと関連づけられている ImqProcess オブジェクト，および ImqQueue オブジェクトはすべて，プログラムの切断前にクローズされ，それぞれの connection reference は切断されます。同一のキューマネジャに複数の ImqQueueManager オブジェクトが接続されている場合，最後に切断されるオブジェクトが物理的な切断を実行します。それ以外のオブジェクトは論理的な切断を実行します。物理的な切断時には，コミットされていない変更内容はコミットされます。正常に実行された場合は TRUE を返します。

MQDISC 命令に相当します。

● ImqBoolean distributionLists(MQLONG & support);

distribution lists 値のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_DIST_LISTS セレクタに相当します。

● MQLONG distributionLists();

考えられるエラーを指示しないで，distribution lists 値を返します。

MQINQ 命令の，MQIA_DIST_LISTS セレクタに相当します。

● ImqObject * firstManagedObject( ) const ;

first managed object を返します。

● ImqBoolean maximumHandles(MQLONG & number);

maximum handles のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_MAX_HANDLES セレクタに相当します。

● MQLONG maximumHandles( );

考えられるエラーを指示しないで，maximum handles を返します。

MQINQ 命令の，MQIA_MAX_HANDLES セレクタに相当します。

● ImqBoolean maximumMessageLength(MQLONG & length);

maximum message length のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_MAX_MSG_LENGTH セレクタに相当します。

● MQLONG maximumMessageLength( );

考えられるエラーを指示しないで，maximum message length を返します。

MQINQ 命令の，MQIA_MAX_MSG_LENGTH セレクタに相当します。

● ImqBoolean maximumPriority(MQLONG & priority);

maximum priority のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_MAX_PRIORITY セレクタに相当します。

● MQLONG maximumPriority( );

考えられるエラーを指示しないで，maximum priority のコピーを返します。

MQINQ 命令の，MQIA_MAX_PRIORITY セレクタに相当します。

● ImqBoolean maximumUncommittedMessages(MQLONG & number);

maximum uncommitted messages のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_MAX_UNCOMMITTED_MSGS セレクタに相当します。

● MQLONG maximumUncommittedMessages( );

考えられるエラーを指示しないで，maximum uncommitted messages を返します。

MQINQ 命令の，MQIA_MAX_UNCOMMITTED_MSGS セレクタに相当します。

● ImqBoolean platform(MQLONG & platform);

platform のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_PLATFORM セレクタに相当します。

● MQLONG platform( );

考えられるエラーを指示しないで，platform を返します。

MQINQ 命令の，MQIA_PLATFORM セレクタに相当します。

- ImqBoolean repositoryName(ImqString & name);

repository name のコピーを提供します。正常に終了した場合は TRUE を返します。

MQINQ 命令の，MQCA_REPOSITORY_NAME セレクタに相当します。

- ImqString repositoryName();

考えられるエラーを指示しないで，repository name を返します。

MQINQ 命令の，MQCA_REPOSITORY_NAME セレクタに相当します。

- ImqBoolean syncPointAvailability(MQLONG & sync);

sync-point availability 値のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_SYNCPOINT セレクタに相当します。

- MQLONG syncPointAvailability();

考えられるエラーを指示しないで，sync-point availability 値のコピーを返します。

MQINQ 命令の，MQIA_SYNCPOINT セレクタに相当します。

- ImqBoolean triggerInterval(MQLONG & interval);

trigger interval のコピーを提供します。正常に実行された場合は TRUE を返します。

MQINQ 命令の，MQIA_TRIGGER_INTERVAL セレクタに相当します。

- MQLONG triggerInterval( );

考えられるエラーを指示しないで，trigger interval を返します。

MQINQ 命令の，MQIA_TRIGGER_INTERVAL セレクタに相当します。

メソッド（protected）

- void setFirstManagedObject(const ImqObject * object = 0);

first managed object を設定します。

オブジェクト属性（protected）

- MQHCONN ohconn

TP1/Message Queue - Access 接続ハンドルです。connection status が TRUE である場合だけ意味を持ちます。

その他の関連クラス

ImqObject

理由コード

- （MQBACK からの理由コード）
- （MQBEGIN からの理由コード）
- （MQCMIT からの理由コード）
- （MQDISC からの理由コード）

理由コードの詳細については，「付録 A　理由コード」，またはマニュアル「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。

ImqReferenceHeader クラス（C++）

このクラスは，MQRMH 構造体の特定の機能をカプセル化します。

インクルードファイル

このクラスを使用するときは，imqrfh.hpp ファイルをインクルードしてください。

オブジェクト属性

● destination environment

宛先の環境です。初期値はヌル文字です。

● destination name

データ宛先の名前です。初期値はヌル文字です。

● instance id

長さが MQ_OBJECT_INSTANCE_ID_LENGTH の 2 進値（MQBYTE24）です。初期値は MQOII_NONE です。

MQRMH 構造体の，ObjectInstanceId フィールドに相当します。

● logical length

このヘッダに続くメッセージデータの論理上の長さ，つまり使用する予定の長さです。初期値は 0 です。

MQRMH 構造体の，DataLogicalLength フィールドに相当します。

● logical offset

後続のメッセージデータの論理オフセットです。最終宛先で，全体としてデータのコンテキストで解釈されます。初期値は 0 です。

MQRMH 構造体の，DataLogicalOffset フィールドに相当します。

● logical offset2

logical offset への高位拡張です。初期値は 0 です。

MQRMH 構造体の，DataLogicalOffset2 フィールドに相当します。

● reference type

参照タイプです。初期値はヌル文字です。

MQRMH 構造体の，ObjectType フィールドに相当します。

- source environment

データ送信側の環境です。初期値はヌル文字です。

- source name

データ送信側の名前です。初期値はヌル文字です。

コンストラクタ

- ImqReferenceHeader();

デフォルトのコンストラクタです。

- ImqReferenceHeader(const ImqReferenceHeader & header);

コピーコンストラクタです。

メソッド

- virtual ImqBoolean copyOut(ImqMessage & msg);

多重定義された ImqItem メソッドです。MQRMH 構造体をメッセージバッファの始めに挿入して，既存のメッセージデータを後ろにずらします。msg format を MQFMT_REF_MSG_HEADER に設定します。

詳細については，「ImqHeader クラス（C++）」を参照してください。

- virtual ImqBoolean pasteIn(ImqMessage & msg);

多重定義された ImqItem メソッドです。メッセージバッファから MQRMH 構造体を読み取ります。

正常に実行するためには，ImqMessage format が MQFMT_REF_MSG_HEADER でなければなりません。

詳細については，「ImqHeader クラス（C++）」を参照してください。

- void operator = (const ImqReferenceHeader & header);

インスタンスデータがヘッダファイルからコピーされ，既存のインスタンスデータと置き換えられます。

- ImqString destinationEnvironment() const;

destination environment のコピーを返します。

- void setDestinationEnvironment(const char * environment = 0);

destination environment を設定します。

- ImqString destinationName() const ;

destination name のコピーを返します。

- void setDestinationName(const char * name = 0);

destination name を設定します。

- ImqBinary instanceId() const ;

instance id のコピーを返します。

- ImqBoolean setInstanceId(const ImqBinary & id);

instance id を設定します。token の data length は，0 または MQ_OBJECT_INSTANCE_ID_LENGTH のどちらかでなければなりません。正常に実行された場合は TRUE を返します。

- void setInstanceId(const MQBYTE24 id = 0);

instance id を設定します。id が 0 の場合は，MQOII_NONE を指定するのと同じです。id が 0 でない場合は，MQ_OBJECT_INSTANCE_ID_LENGTH バイトの 2 進データをアドレス指定する必要があります。

MQOII_NONE などの事前定義値を使用する場合は，確実に信号機能が一致するようにキャストを作成する必要があります。

キャストは，例えば (MQBYTE *)MQOII_NONE のように作成します。

- MQLONG logicalLength() const ;

logical length を返します。

- void setLogicalLength(const MQLONG length);

logical length を設定します。

- MQLONG logicalOffset() const ;

logical offset を返します。

- void setLogicalOffset(const MQLONG offset);

logical offset を設定します。

- MQLONG logicalOffset2() const;

logical offset2 を返します。

- void setLogicalOffset2(const MQLONG offset);

logical offset2 を設定します。

- ImqString referenceType() const ;

reference type のコピーを返します。

- void setReferenceType(const char * name = 0);

reference type を設定します。

- ImqString sourceEnvironment() const ;

source environment のコピーを返します。

- void setSourceEnvironment(const char * environment = 0);

source environment を設定します。

- ImqString sourceName() const ;

source name のコピーを返します。

- void setSourceName(const char * name = 0);

source name を設定します。

オブジェクト属性（protected）

- MQRMH omqrmh

MQRMH 構造体です。

その他の関連クラス

ImqBinary

ImqHeader

ImqItem

ImqMessage

ImqString

ImqString クラス（C++）

このクラスは，ヌル文字で終了するストリングに文字ストリング記憶域と操作を提供します。パラメタが char* を呼び出す状態では，多くの場合，char* の代わりに ImqString を使用できます。

インクルードファイル

このクラスを使用するときは，imqstr.hpp ファイルをインクルードしてください。

オブジェクト属性

- characters

ヌル文字の前にある storage 内の文字です。

- length

characters 内のバイト数です。storage がない場合，length は 0 です。初期値は 0 です。

- storage

任意のサイズの，バイトアレイです。characters の終わりを検出できるように，characters の後ろの storage には，必ずヌル文字が含まれなければなりません。メソッドは，この状態が必ず保持されるようにしますが，アレイにバイトを直接設定するときには，必ず，変更の後ろにヌル文字があるように注意する必要があります。最初は，storage はありません。

コンストラクタ

- ImqString( );

デフォルトのコンストラクタです。

- ImqString(const ImqString & string);

コピーコンストラクタです。

- ImqString(const char c);

characters は c から構成されます。

- ImqString(const char * text);

characters は text からコピーされます。

- ImqString(const void * buffer，const size_t length);

buffer から length 分のバイトをコピーし，それらを characters に割り当てます。コピーされたすべてのヌル文字は，ピリオド（.）に置き換えます。ヌル文字以外の印刷不能ま

たは表示不能文字はそのままになります。

メソッド

● static ImqBoolean copy(char * destination-buffer，const size_t length，const char* source-buffer，const char pad = 0);

最大 length 分のバイトを source-buffer から destination-buffer へコピーします。

source-buffer 内の文字数が不十分な場合，destination-buffer 内の残りのスペースに pad 文字が埋められます。source-buffer は 0 でもかまいません。length も 0 の場合は，destination-buffer はヌルでかまいません。正常に実行された場合は TRUE を返します。

● virtual ImqBoolean copyOut(ImqMessage & msg);

多重定義された ImqItem メソッドです。characters をメッセージバッファへコピーし，既存の内容と置き換えます。msgformat を MQFMT_STRING に設定します。

詳細については，親クラスのメソッドの説明を参照してください。

● virtual ImqBoolean pasteIn(ImqMessage & msg);

多重定義された ImqItem メソッドです。メッセージバッファから残りのデータを転送することによって characters を設定し，既存の characters を置き換えます。

正常に実行されるためには，次の指定をしなければなりません。

- msg オブジェクトの encoding が MQENC_NATIVE でなければなりません。メッセージは，MQGMO_CONVERT を MQENC_NATIVE に設定して検索してください。
- ImqMessage format が MQFMT_STRING でなければなりません。

詳細については，親クラスのメソッドの説明を参照してください。

● char & operator [ ] (const size_t offset) const ;

storage 内のオフセット（offset）にある文字を参照します。必ず関係のあるバイトが存在し，アドレス可能であるようにしてください。

● ImqString operator ( ) (const size_t offset，const size_t length = 1) const ;

offset から始まる characters からバイトをコピーすることによってサブストリングを返します。

length が 0 の場合は，characters の残りが返されます。offset と length を組み合わせても characters 内で参照されない場合には，空の ImqString が返されます。

● void operator = (const ImqString & string);

インスタンスデータが string からコピーされ，既存のインスタンスデータと置き換えられます。

- ImqString operator + (const char c) const ;

c を characters に付加した結果を返します。

- ImqString operator + (const char * text) const ;

text を characters に付加した結果を返します。位置を逆に指定できます。例えば，次のどちらの形式で指定してもかまいません。

```
strOne + "string two" ;
"string one" + strTwo;
```

- ImqString operator + (const ImqString & string1) const ;

string1 を characters に付加した結果を返します。

- ImqString operator + (const double number) const ;

テキストに変換後 number を characters に付加した結果を返します。

- ImqString operator + (const long number) const ;

テキストに変換後 number を characters に付加した結果を返します。

- void operator += (const char c);

c は characters に付加されます。

- void operator += (const char * text);

text を characters に付加します。

- void operator += (const ImqString & string);

string を characters に付加します。

- void operator += (const double number);

テキストに変換後 number を characters に付加します。

- void operator += (const long number);

テキストに変換後 number を characters に付加します。

- void operator char * ( ) const ;

storage 内の最初のバイトのアドレスを返します。この値は 0 でもかまいません。

- ImqBoolean operator < (const ImqString & string) const ;
- ImqBoolean operator > (const ImqString & string) const ;
- ImqBoolean operator <= (const ImqString & string) const ;
- ImqBoolean operator >= (const ImqString & string) const ;
- ImqBoolean operator == (const ImqString & string) const ;
- ImqBoolean operator != (const ImqString & string) const ;

compare メソッドを使用して，characters を string の内容と比較します。TRUE または FALSE のどれかを返します。

- short compare(const ImqString & string) const ;

characters を string の内容と比較します。結果は，両方の characters が等しい場合は 0，前者の方が小さい場合は負，大きい場合は正です。比較には，大文字小文字の区別があります。

空の ImqString は，空でない ImqString より小さいと見なされます。

- ImqBoolean copyOut(char * buffer，const size_t length，const char pad = 0);

最大 length 分のバイトを characters から buffer へコピーします。buffer 内の characters の数が不十分な場合，buffer 内の残りのスペースに pad 文字が埋められます。length も 0 であれば，buffer は 0 でもかまいません。正常に実行された場合は TRUE を返します。

- size_t copyOut(long & number) const ;

テキストから変換後，characters から number を設定します。変換にかかわった文字数を返します。

これが 0 の場合，変換されていないので，number は設定されません。変換文字シーケンスは，次の形式で始まっている必要があります。

ストリングテキストから整数への変換の形式は次のとおりです。

```
<blank(s)>
<+|->
digit(s)
```

- size_t copyOut(ImqString & token，const char c = ' ') const ;

characters に c と異なる一つまたは複数の文字が含まれている場合，token は，そのような文字の最初の連続するシーケンスとして識別されます。この場合，token はそのシーケンスに設定され，返された値は先行文字 c の数とシーケンス中のバイトの数との合計です。それ以外の場合は，0 が返され，token は設定されません。

● size_t cutOut(long & number);

copy メソッドの場合と同様に number を設定しますが，さらに，戻り値によって指示されたバイト数を characters から除去します。例えば，次に示すストリングは，cutOut(number) を 3 回使用することによって，三つの数値に分割できます。

ストリングテキストから整数を検索する例を次に示します。

```
strNumbers = "-1 0     +55 ";
  while ( strNumbers.cutOut( number ) );
  number becomes -1, then 0, then 55
  leaving strNumbers == " "
```

● size_t cutOut(ImqString & token, const char c = ' ');

copyOut メソッドの場合と同様に token を設定し，characters から strToken 分の文字を除去し，さらに，token 分の文字の前にある任意の文字 c も除去します。c が空白でない場合，token 分の文字の直後に続く文字 c も除去されます。除去された文字の数を返します。例えば，次のストリングは，cutOut(token) を 3 回使用することによって，三つの token に分割できます。

ストリングテキストから token を検索する例を次に示します。

```
  strText = "  Program Version 1.1  ";
   while ( strText.cutOut( token ) );
  // token becomes "Program", then "Version",
  // then "1.1" leaving strText == "  "
```

ストリング内の DOS パス名を解析する方法について，次の例で示します。

```
  strPath = "C:¥¥WINNT¥¥WINNT.BMP"
  strPath.cutOut( strDrive, ':' );
  strPath.stripLeading( ':' );
  while ( strPath.cutOut( strFile, '¥¥' ) );
  // strDrive becomes "C".
  // strFile becomes "WINNT",
  // then "WINNT.BMP" leaving strPath empty.
```

● ImqBoolean find(const ImqString & string);

characters 内のどこかに string の完全一致がないか探索します。一致が見つからない場合は，FALSE を返します。一致が見つかった場合は，TRUE を返します。string が空の場合は，TRUE を返します。

● ImqBoolean find(const ImqString & string , size_t & offset);

オフセット offset から先の characters 内のどこかに string の完全一致がないか探索します。

string が空の場合は，offset を更新しないで TRUE を返します。一致が見つからない場合は FALSE を返します。offset の値が増やされている場合もあるので注意してください。一致が見つかった場合は，TRUE を返し，offset を characters 内の string のオフセットに更新します。

- size_t length( ) const ;

length を返します。

- ImqBoolean pasteIn(const double number，const char * format = "%f");

テキストへの変換後，number が characters に付加されます。正常に実行された場合は TRUE を返します。

浮動小数点変換を形式設定するために，指定 format が使用されます。これを指定する場合は，printf，および浮動小数点数（例えば，"%.3f"）とともに使用するのに適したものでなければなりません。

- ImqBoolean pasteIn(const long number);

テキストへの変換後，number が characters に付加されます。正常に実行された場合は TRUE を返します。

- ImqBoolean pasteIn(const void * buffer，const size_t length);

buffer から characters へ length 分のバイトを付加し，最後のヌル文字を追加します。コピーされたあらゆるヌル文字に対して置き換えられます。置き換えられる文字はピリオド（.）です。コピーされたその他の印刷不能または表示不能文字に特別な考慮はされません。正常に実行された場合は TRUE を返します。

- ImqBoolean set(const char * buffer，const size_t length);

固定長文字フィールドからの characters を設定します。これには，ヌル文字が含まれている場合もありますが，含まれていない場合もあります。必要に応じて固定長フィールドからの文字にヌル文字が付加されます。正常に実行された場合は TRUE を返します。

- size_t storage( ) const ;

storage 内のバイト数を返します。

- ImqBoolean setStorage(const size_t length);

storage を（再度）割り振り，現在割り振られているバイトの数を返します。ヌル文字を含め，元の characters が収まるだけの余裕がある場合には，それらは保存されますが，追加の記憶域は初期化されません。

正常に実行された場合は TRUE を返します。

● size_t stripLeading(const char c = ' ');

characters から先行文字 c を除去し，除去された数を返します。

● size_t stripTrailing(const char c = ' ');

characters から文字 c を除去し，除去された数を返します。

● ImqString upperCase( ) const ;

characters の大文字のコピーを返します。

メソッド（protected）

● ImqBoolean assign(const ImqString & string);

同等の operator= メソッドに等しい非仮想のメソッドです。

正常に実行された場合は TRUE を返します。

その他の関連クラス

ImqItem

ImqMessage

注意

ほとんどのコンパイラは strOne + "string two" を受け入れますが，Visual C++ では strOne + (char *)"string two" が必要です。

理由コード

- MQRC_DATA_TRUNCATED
- MQRC_NULL_POINTER
- MQRC_STORAGE_NOT_AVAILABLE

理由コードの詳細については，「付録 A　理由コード」，またはマニュアル「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。

ImqTrigger クラス（C++）

このクラスは，MQTM 構造体をカプセル化します。このクラスのオブジェクトは，通常，トリガモニタプログラムが使用します。このプログラムの機能は特定のメッセージを待ち，メッセージに加工を加えることによって，メッセージがほかの TP1/Message Queue - Access アプリケーションを待っているとき，これらのアプリケーションの起動を確実にすることです。

インクルードファイル

このクラスを使用するとき，imqtrg.hpp ファイルをインクルードしてください。

オブジェクト属性

- application id

該当するメッセージを送信したアプリケーションの ID です。初期値はヌル文字です。

MQTM 構造体の，ApplId フィールドに相当します。

- application type

該当するメッセージを送信したアプリケーションのタイプです。初期値は 0 です。

MQTM 構造体の，ApplType フィールドに相当します。

- environment data

プロセスの環境データです。初期値はヌル文字です。

MQTM 構造体の，EnvData フィールドに相当します。

- process name

プロセス名です。初期値はヌル文字です。

MQTM 構造体の，ProcessName フィールドに相当します。

- queue name

開始されるキューの名前です。初期値はヌル文字です。

MQTM 構造体の，QName フィールドに相当します。

- trigger data

プロセスのトリガデータです。初期値はヌル文字です。

MQTM 構造体の，TriggerData フィールドに相当します。

- user data

プロセスのユーザデータです。初期値はヌル文字です。

MQTM 構造体の，UserData フィールドに相当します。

コンストラクタ

- ImqTrigger( );

デフォルトのコンストラクタです。

- ImqTrigger(const ImqTrigger & trigger);

コピーコンストラクタです。

メソッド

- virtual ImqBoolean copyOut(ImqMessage & msg);

多重定義された ImqItem メソッドです。MQTM 構造体をメッセージバッファに書き込み，既存の内容と置き換えます。msg format を MQFMT_TRIGGER に設定します。

詳細については，「ImqItem クラス（C++）」の「メソッド」を参照してください。

- virtual ImqBoolean pasteIn(ImqMessage & msg);

多重定義された ImqItem メソッドです。メッセージバッファから MQTM 構造体を読み取ります。

正常に実行されるためには，ImqMessage format が MQFMT_TRIGGER でなければなりません。

詳細については，「ImqItem クラス（C++）」の「メソッド」を参照してください。

- void operator =(const ImqTrigger & trigger);

インスタンスデータが trigger からコピーされ，既存のインスタンスデータと置き換えられます。

- ImqString applicationId( ) const ;

application id のコピーを返します。

- void setApplicationId(const char * id);

application id を設定します。

- MQLONG applicationType( ) const ;

application type を返します。

- void setApplicationType(const MQLONG type);

application type を設定します。

- ImqBoolean copyOut(MQTMC2 * ptmc2);

このクラスは，開始キューで受信された MQTM 構造体をカプセル化します。このメソッドは，呼び出し側によって提供された同等な MQTMC2 構造体を設定し，QMgrName フィールドをすべて空白に設定します。なお，QMgrName フィールドは MQTM 構造体には存在しません。

MQTMC2 構造体は，従来，トリガモニタによって開始されたアプリケーションに対するパラメタとして使用されています。正常に実行された場合は TRUE を返します。

- ImqString environmentData( ) const ;

environment data のコピーを返します。

- void setEnvironmentData(const char * data);

environment data を設定します。

- ImqString processName( ) const ;

process name のコピーを返します。

- void setProcessName(const char * name);

process name を設定します。

- ImqString queueName( ) const ;

queue name のコピーを返します。

- void setQueueName(const char * name);

queue name を設定します。

- ImqString triggerData( ) const ;

trigger data のコピーを返します。

- void setTriggerData(const char * data);

trigger data を設定します。

- ImqString userData( ) const ;

user data のコピーを返します。

- void setUserData(const char * data);

user data を設定します。

オブジェクト属性（protected）

- MQTM omqtm

MQTM 構造体です。

その他の関連クラス

ImqGetMessageOptions

ImqItem

ImqMessage

ImqString

理由コード

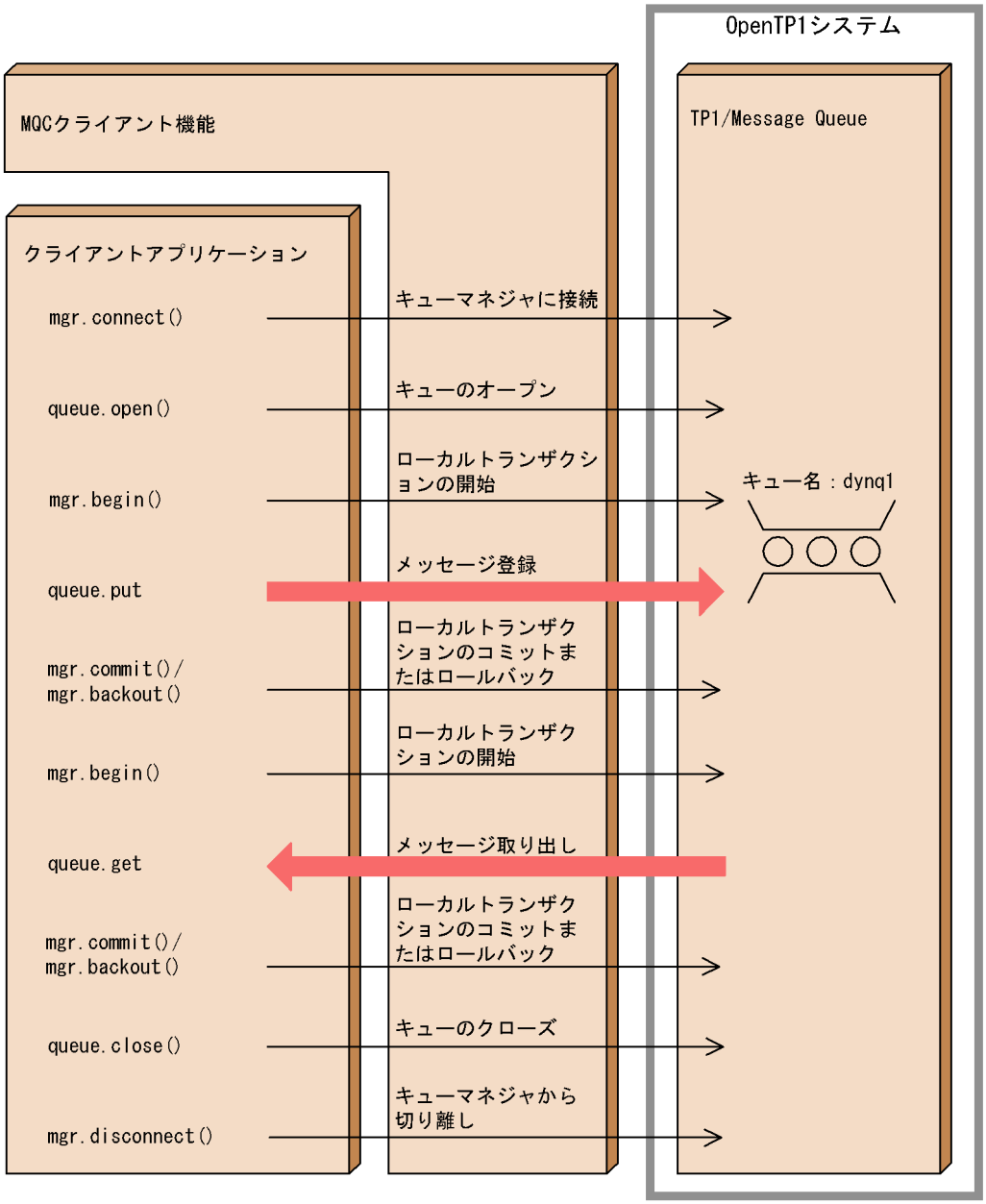
- MQRC_NULL_POINTER

理由コードの詳細については,「付録 A　理由コード」を参照してください。

C++ のサンプルアプリケーション

C++ のサンプルアプリケーションです。処理の流れを次の図に示します。

図 5-2　処理の流れ（C++ の場合）



C++ のサンプルコーディング

コーディング例を示します。

```
#include <stdio.h>
#include <imqi.hpp>          // include MQAccess C++ class

int main (void) {
  ImqQueueManager  mgr(" ");      /* queue manager name */
  ImqQueue       queue;           /* queue */
  ImqMessage     msg;             /* message */
  ImqString      str("****** C++ sample put data ******");
                                  /* message data */
  ImqGetMessageOptions getoption; /* get message options */
  ImqPutMessageOptions putoption; /* put message options */

  // connect to the queue manager
  if ( ! mgr.connect( ) ) {
    /* stop if it failed */
    printf(
    "ImqQueueManager::connect ended with reason code %d¥n",
                                  (int)mgr.reasonCode( ) );
    return ( (int)mgr.reasonCode( ) );
  }

  // set the connection reference
  queue.setConnectionReference( mgr );

  // set the queue name
  queue.setName("dynq1");

  // set the message
  msg.writeItem( str );

  // set the open options
  queue.setOpenOptions(MQOO_OUTPUT | MQOO_INPUT_AS_Q_DEF);

  // open the queue
  queue.open( );
  if ( queue.reasonCode( ) ) {
    printf( "ImqQueue::open ended with reason code %d¥n",
                               (int)queue.reasonCode( ) );
    return( (int)queue.reasonCode( ) );
  }

  // set the get message options
  getoption.setSyncPointParticipation(TRUE);
  // set the put message options
  putoption.setSyncPointParticipation(TRUE);

  // begin local transaction
  if ( ! mgr.begin( ) ) {
    printf(
    "ImqQueueManager::begin ended with reason code %d¥n",
                                  (int)mgr.reasonCode( ) );
```

```
  return( (int)mgr.reasonCode( ) );
}

// put the message
if ( ! queue.put( msg, putoption ) ) {
  printf( "ImqQueue::put ended with reason code %d¥n",
                                (int)queue.reasonCode( ) );
}

// commit local transaction
if ( queue.reasonCode( ) == 0 ) {
  if ( ! mgr.commit( ) ) {
    printf(
   "ImqQueueManager::commit ended with reason code %d¥n",
                                  (int)mgr.reasonCode( ) );
    return( (int)mgr.reasonCode( ) );
  }
} else {
  if ( ! mgr.backout( ) ) {
    printf(
  "ImqQueueManager::backout ended with reason code %d¥n",
                                  (int)mgr.reasonCode( ) );
    return( (int)mgr.reasonCode( ) );
  }
}

// begin local transaction
if ( ! mgr.begin( ) ) {
  printf(
   "ImqQueueManager::begin ended with reason code %d¥n",
                                  (int)mgr.reasonCode( ) );
  return( (int)mgr.reasonCode( ) );
}

// get the message
if ( ! queue.get( msg, getoption ) ) {
  printf( "ImqQueue::get ended with reason code %d¥n",
                                (int)queue.reasonCode( ) );
}

// commit local transaction
if ( queue.reasonCode( ) == 0 ) {
  if ( ! mgr.commit( ) ) {
    printf(
   "ImqQueueManager::commit ended with reason code %d¥n",
                                  (int)mgr.reasonCode( ) );
    return( (int)mgr.reasonCode( ) );
  }
} else {
  if ( ! mgr.backout( ) ) {
    printf(
  "ImqQueueManager::backout ended with reason code %d¥n",
                                  (int)mgr.reasonCode( ) );
    return( (int)mgr.reasonCode( ) );
  }
}

// close the queue
```

```
    if ( ! queue.close( ) ) {
      printf( "ImqQueue::close ended with reason code %d¥n",
                                  (int)queue.reasonCode( ) );
    }

    // disconnect from the queue manager
    if ( ! mgr.disconnect( ) ) {
      printf(
   "ImqQueueManager::disconnect ended with reason code %d¥n",
                                    (int)mgr.reasonCode( ) );
    }
    return (0);
  }
```

6 MQC クライアント機能の Java インタフェース

この章では，MQC クライアント機能の Java インタフェースについて説明します。

MQC クライアント機能の Java パッケージ

MQC クライアント機能の Java クラス一覧

Java クラス継承図

MQDistributionList クラス（Java）

MQDistributionListItem クラス（Java）

MQEnvironment クラス（Java）

MQException クラス（Java）

MQGetMessageOptions クラス（Java）

MQManagedObject クラス（Java）

MQMessage クラス（Java）

MQMessageTracker クラス（Java）

MQProcess クラス（Java）

MQPutMessageOptions クラス（Java）

MQQueue クラス（Java）

MQQueueManager クラス（Java）

MQC インタフェース（Java）

Java のサンプルアプリケーション

Java のサンプルコーディング

MQC クライアント機能の Java パッケージ

MQC クライアント機能の Java インタフェースは，パッケージとして次の非 XA インタフェースを提供します。

- JP.co.Hitachi.soft.MQ.Access
  非 XA インタフェースパッケージです。ローカルトランザクション機能だけ使用できます。

MQC クライアント機能の Java クラス一覧

MQC クライアント機能が提供するクラスの概要について説明します。

クラスの一覧を次の表に示します。

表 6-1 クラスの一覧（Java）

クラス名	機能
MQDistributionList	配布リストです。
MQDistributionListItem	配布リスト中の単一項目です。
MQEnvironment	Java 環境を設定します。
MQException	MQ の例外発生を示します。
MQGetMessageOptions	MQQueue:get メソッドのオプションです。
MQManagedObject	MQQueueManager，MQQueue，MQProcess のスーパクラスです。
MQMessage	MQ で登録・取り出しするメッセージです。
MQMessageTracker	メッセージパラメタです。
MQProcess	MQ のプロセス定義にアクセスします。
MQPutMessageOptions	MQQueue:put メソッドのオプションです。
MQQueue	MQ のキューにアクセスします。
MQQueueManager	MQ のキューマネジャにアクセスします。

インタフェースの一覧を次の表に示します。

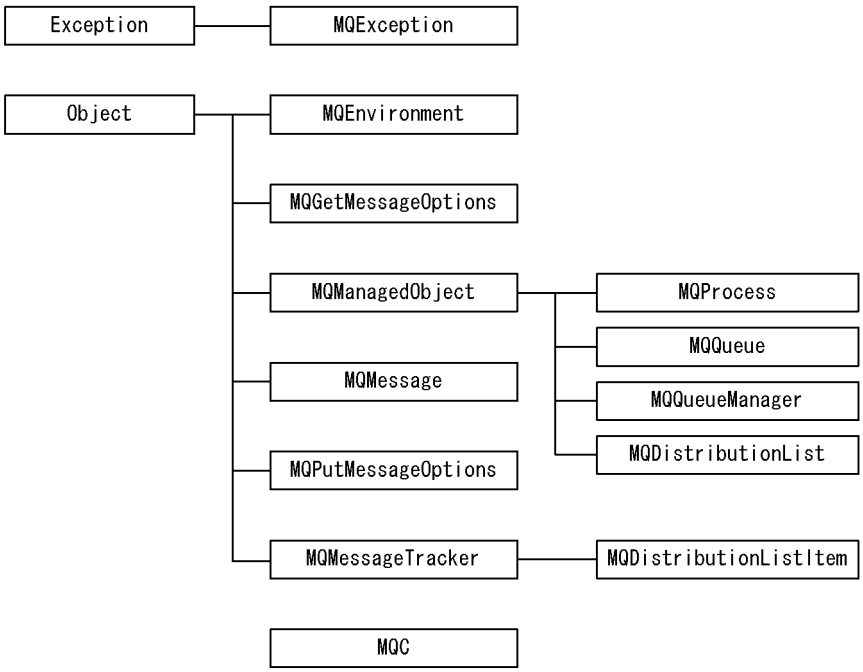
表 6-2 インタフェースの一覧（Java）

インタフェース名	機能
MQC	TP1/Message Queue の定義値です。

Java クラス継承図

クラス継承図を次の図に示します。

図 6-1 クラス継承図（Java の場合）



（凡例）

――――― ：継承します。

MQDistributionList クラス（Java）

このクラスは，配布リストです。

コンストラクタ

● public MQDistributionList(MQQueueManager qMgr, MQDistributionListItem[] litems, Int openOptions, String alternateUserId) throws MQException

MQDistributionList コンスタラクタです。

配布リストは，put() メソッドへの単一呼び出しを使用してメッセージを送信できる一連のオープンキューを表示します。

パラメタ

qMgr
　配布リストがオープンされるキューマネジャを指定します。

litems
　配布リストに組み込む項目を指定します。

openOptions
　オープンオプションを指定します。

alternateUserId
　代替ユーザ識別子を指定します。

メソッド

● public synchronized void put(MQMessage message, MQPutMessageOptions putMessageOptions) throws MQException

メッセージを配布リスト上のキューに書き込みます。

配布リスト用 MQPUT 命令に相当します。

パラメタ

message
　メッセージ記述子情報，および戻されたメッセージデータを格納する入出力パラメタを指定します。

putMessageOptions
　MQPUT 命令の動作を制御するオプションを指定します。

● public MQDistributionListItem getFirstDistributionListItem()

配布リストの先頭項目を戻します。配布リストが空の場合は null を戻します。

- public int getValidDestinationCount()

正常にオープンされた配布リストの項目数を戻します。

MQPUT 命令の，MQPMO 構造体の KnownDestCount と UnknownDestCount の合計に相当します。

- public int getInvalidDestinationCount()

正常にオープンされなかった配布リストの項目数を戻します。

MQPUT 命令の，MQPMO 構造体の InvalidDestCount に相当します。

MQDistributionListItem クラス（Java）

このクラスは，配布リスト中の単一項目です。

変数

- public int completionCode

完了コードです。

- public String queueName

配布リストで使用するキューの名前です。

- public String queueManagerName

キューが定義されているキューマネジャの名前です。

- public int reasonCode

理由コードです。

コンストラクタ

- public MQDistributionListItem()

MQDistributionListItem コンストラクタです。

配布リスト中の単一項目（キュー）を表示します。

MQEnvironment クラス（Java）

このクラスでは，Java 環境の設定をします。

変数

● public final static String version_notice

MQC クライアント機能の Java インタフェースバージョンです。

メソッド

● public static void disableTracing()

トレース機能を停止します。

● public static void enableTracing(int level)

JavaEnvironment トレース機能を開始します。JavaEnvironment トレース情報は，Java コンソール（System.err）に出力されます。出力形式については，「2.4.2 JavaEnvironment トレースファイルの出力形式」を参照してください。

パラメタ

level

JavaEnvironment トレース情報の出力レベルを 1 ～ 5 の値で指定します。指定した値に応じて，次の表に示すトレース情報が出力されます。

表 6-3 JavaEnvironment トレース情報の出力レベル

出力情報	出力レベル				
	1	2	3	4	5
記入項目	○	○	○	○	○
出口情報	○	○	○	○	○
例外トレース	○	○	○	○	○
入口情報	×	○	○	○	○
パラメタ情報	×	×	○	○	○
送信／受信の MQ ヘッダ	×	×	×	○	○
MQGMO 情報／ MQPMO 情報	×	×	×	○	○
送信／受信ユーザメッセージデータ	×	×	×	×	○

（凡例）
○：出力されます。
×：出力されません。

● public static void enableTracing(int level, OutputStream stream)

トレース機能を開始します。トレース情報の出力先を指定できます。

パラメタ

level

トレース出力レベルを 1 ～ 5 の値で指定します。各レベルについては，上記の enableTracing メソッドを参照してください。

stream

トレース情報の出力先を指定します。

コンストラクタ

● public MQEnvironment()

MQEnvironment コンストラクタです。

MQException クラス（Java）

このクラスでは，MQ の例外発生を管理します。

変数

● public static java.io.outputStreamWriter log

例外ログ出力先を指定します。省略すると System.err に出力します。また，null を設定するとログを出力しません。

● public Object exceptionSource

例外発生オブジェクトです。

● public int completionCode

完了コードです。完了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。

● public int reasonCode

理由コードです。理由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。

完了コード，理由コード

完了コード，理由コードの値の一覧を示します。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。

- public static final int MQCC_WARNING
- public static final int MQCC_FAILED
- public static final int MQRC_ADAPTER_STORAGE_SHORTAGE
- public static final int MQRC_ADAPTER_CONN_LOAD_ERROR
- public static final int MQRC_ADAPTER_SERV_LOAD_ERROR
- public static final int MQRC_ADAPTER_DEFS_ERROR
- public static final int MQRC_ADAPTER_DEFS_LOAD_ERROR
- public static final int MQRC_ADAPTER_CONV_LOAD_ERROR
- public static final int MQRC_ADAPTER_DISC_LOAD_ERROR
- public static final int MQRC_ADAPTER_NOT_AVAILABLE
- public static final int MQRC_ALIAS_BASE_Q_TYPE_ERROR
- public static final int MQRC_ALREADY_CONNECTED
- public static final int MQRC_ANOTHER_Q_MGR_CONNECTED
- public static final int MQRC_API_EXIT_LOAD_ERROR
- public static final int MQRC_ASID_MISMATCH
- public static final int MQRC_BACKED_OUT

- public static final int MQRC_BO_ERROR
- public static final int MQRC_BRIDGE_STARTED
- public static final int MQRC_BRIDGE_STOPPED
- public static final int MQRC_BUFFER_ERROR
- public static final int MQRC_BUFFER_LENGTH_ERROR
- public static final int MQRC_CALL_IN_PROGRESS
- public static final int MQRC_CFH_ERROR
- public static final int MQRC_CFIL_ERROR
- public static final int MQRC_CFIN_ERROR
- public static final int MQRC_CFSL_ERROR
- public static final int MQRC_CFST_ERROR
- public static final int MQRC_CHANNEL_AUTO_DEF_ERROR
- public static final int MQRC_CHANNEL_AUTO_DEF_OK
- public static final int MQRC_CHANNEL_ACTIVATED
- public static final int MQRC_CHANNEL_NOT_ACTIVATED
- public static final int MQRC_CHANNEL_STARTED
- public static final int MQRC_CHANNEL_STOPPED
- public static final int MQRC_CHANNEL_CONV_ERROR
- public static final int MQRC_CICS_WAIT_FAILED
- public static final int MQRC_CONNECTION_QUIESCING
- public static final int MQRC_CONNECTION_STOPPING
- public static final int MQRC_CONNECTION_NOT_AUTHORIZED
- public static final int MQRC_CORREL_ID_ERROR
- public static final int MQRC_CHAR_ATTR_LENGTH_ERROR
- public static final int MQRC_CHAR_ATTRS_ERROR
- public static final int MQRC_CHAR_ATTRS_TOO_SHORT
- public static final int MQRC_CLUSTER_PUT_INHIBITED
- public static final int MQRC_CLUSTER_RESOLUTION_ERROR
- public static final int MQRC_CONTEXT_HANDLE_ERROR
- public static final int MQRC_CONTEXT_NOT_AVAILABLE
- public static final int MQRC_COD_NOT_VALID_FOR_XCF_Q
- public static final int MQRC_CONVERTED_MSG_TOO_BIG
- public static final int MQRC_CNO_ERROR
- public static final int MQRC_CONN_ID_IN_USE
- public static final int MQRC_CONNECTION_BROKEN
- public static final int MQRC_DATA_LENGTH_ERROR
- public static final int MQRC_DBCS_ERROR
- public static final int MQRC_DEF_XMIT_Q_TYPE_ERROR
- public static final int MQRC_DEF_XMIT_Q_USAGE_ERROR
- public static final int MQRC_DEST_ENV_ERROR

- public static final int MQRC_DEST_NAME_ERROR
- public static final int MQRC_DH_ERROR
- public static final int MQRC_DLH_ERROR
- public static final int MQRC_DUPLICATE_RECOV_COORD
- public static final int MQRC_DYNAMIC_Q_NAME_ERROR
- public static final int MQRC_ENVIRONMENT_ERROR
- public static final int MQRC_EXPIRY_ERROR
- public static final int MQRC_FEEDBACK_ERROR
- public static final int MQRC_FILE_SYSTEM_ERROR
- public static final int MQRC_FORMAT_ERROR
- public static final int MQRC_FUNCTION_ERROR
- public static final int MQRC_GET_INHIBITED
- public static final int MQRC_GMO_ERROR
- public static final int MQRC_GROUP_ID_ERROR
- public static final int MQRC_HANDLE_NOT_AVAILABLE
- public static final int MQRC_HCONFIG_ERROR
- public static final int MQRC_HCONN_ERROR
- public static final int MQRC_HEADER_ERROR
- public static final int MQRC_HOBJ_ERROR
- public static final int MQRC_INHIBIT_VALUE_ERROR
- public static final int MQRC_IIH_ERROR
- public static final int MQRC_INCOMPLETE_GROUP
- public static final int MQRC_INCOMPLETE_MSG
- public static final int MQRC_INCONSISTENT_BROWSE
- public static final int MQRC_INCONSISTENT_CCSID
- public static final int MQRC_INCONSISTENT_ENCODINGS
- public static final int MQRC_INCONSISTENT_PERSISTENCE
- public static final int MQRC_INCONSISTENT_UOW
- public static final int MQRC_INITIALIZATION_FAILED
- public static final int MQRC_INVALID_MSG_UNDER_CURSOR
- public static final int MQRC_INT_ATTR_COUNT_ERROR
- public static final int MQRC_INT_ATTR_COUNT_TOO_SMALL
- public static final int MQRC_INT_ATTRS_ARRAY_ERROR
- public static final int MQRC_MATCH_OPTIONS_ERROR
- public static final int MQRC_MAX_CONNS_LIMIT_REACHED
- public static final int MQRC_MD_ERROR
- public static final int MQRC_MDE_ERROR
- public static final int MQRC_MISSING_REPLY_TO_Q
- public static final int MQRC_MSG_ID_ERROR
- public static final int MQRC_MSG_FLAGS_ERROR

- public static final int MQRC_MSG_SEQ_NUMBER_ERROR
- public static final int MQRC_MSG_TOO_BIG_FOR_CHANNEL
- public static final int MQRC_MSG_TOO_BIG_FOR_Q
- public static final int MQRC_MSG_TOO_BIG_FOR_Q_MGR
- public static final int MQRC_MSG_TYPE_ERROR
- public static final int MQRC_MULTIPLE_REASONS
- public static final int MQRC_NAME_IN_USE
- public static final int MQRC_NAME_NOT_VALID_FOR_TYPE
- public static final int MQRC_NOT_CONVERTED
- public static final int MQRC_NO_MSG_LOCKED
- public static final int MQRC_NO_DESTINATIONS_AVAILABLE
- public static final int MQRC_NO_EXTERNAL_PARTICIPANTS
- public static final int MQRC_NO_MSG_AVAILABLE
- public static final int MQRC_NO_MSG_UNDER_CURSOR
- public static final int MQRC_NOT_AUTHORIZED
- public static final int MQRC_NOT_OPEN_FOR_BROWSE
- public static final int MQRC_NOT_OPEN_FOR_INPUT
- public static final int MQRC_NOT_OPEN_FOR_INQUIRE
- public static final int MQRC_NOT_OPEN_FOR_OUTPUT
- public static final int MQRC_NOT_OPEN_FOR_SET
- public static final int MQRC_NOT_OPEN_FOR_PASS_ALL
- public static final int MQRC_NOT_OPEN_FOR_PASS_IDENT
- public static final int MQRC_NOT_OPEN_FOR_SET_ALL
- public static final int MQRC_NOT_OPEN_FOR_SET_IDENT
- public static final int MQRC_OBJECT_ALREADY_EXISTS
- public static final int MQRC_OBJECT_CHANGED
- public static final int MQRC_OBJECT_DAMAGED
- public static final int MQRC_OBJECT_IN_USE
- public static final int MQRC_OBJECT_NAME_ERROR
- public static final int MQRC_OBJECT_Q_MGR_NAME_ERROR
- public static final int MQRC_OBJECT_RECORDS_ERROR
- public static final int MQRC_OBJECT_TYPE_ERROR
- public static final int MQRC_OD_ERROR
- public static final int MQRC_OFFSET_ERROR
- public static final int MQRC_OPEN_FAILED
- public static final int MQRC_OPTION_NOT_VALID_FOR_TYPE
- public static final int MQRC_OPTIONS_ERROR
- public static final int MQRC_ORIGINAL_LENGTH_ERROR
- public static final int MQRC_OUTCOME_MIXED
- public static final int MQRC_OUTCOME_PENDING

- public static final int MQRC_PARTICIPANT_NOT_AVAILABLE
- public static final int MQRC_PAGESET_FULL
- public static final int MQRC_PAGESET_ERROR
- public static final int MQRC_PCF_ERROR
- public static final int MQRC_PERSISTENCE_ERROR
- public static final int MQRC_PERSISTENT_NOT_ALLOWED
- public static final int MQRC_PMO_ERROR
- public static final int MQRC_PMO_RECORD_FLAGS_ERROR
- public static final int MQRC_PRIORITY_EXCEEDS_MAXIMUM
- public static final int MQRC_PRIORITY_ERROR
- public static final int MQRC_PUT_INHIBITED
- public static final int MQRC_PUT_MSG_RECORDS_ERROR
- public static final int MQRC_Q_ALREADY_EXISTS
- public static final int MQRC_Q_DEPTH_HIGH
- public static final int MQRC_Q_DEPTH_LOW
- public static final int MQRC_Q_DELETED
- public static final int MQRC_Q_FULL
- public static final int MQRC_Q_MGR_ACTIVE
- public static final int MQRC_Q_MGR_NAME_ERROR
- public static final int MQRC_Q_MGR_NOT_AVAILABLE
- public static final int MQRC_Q_MGR_NOT_ACTIVE
- public static final int MQRC_Q_MGR_QUIESCING
- public static final int MQRC_Q_SERVICE_INTERVAL_HIGH
- public static final int MQRC_Q_SERVICE_INTERVAL_OK
- public static final int MQRC_Q_MGR_STOPPING
- public static final int MQRC_Q_NOT_EMPTY
- public static final int MQRC_Q_SPACE_NOT_AVAILABLE
- public static final int MQRC_Q_TYPE_ERROR
- public static final int MQRC_RECS_PRESENT_ERROR
- public static final int MQRC_REPORT_OPTIONS_ERROR
- public static final int MQRC_RESOURCE_PROBLEM
- public static final int MQRC_RESPONSE_RECORDS_ERROR
- public static final int MQRC_REMOTE_Q_NAME_ERROR
- public static final int MQRC_RMH_ERROR
- public static final int MQRC_SECOND_MARK_NOT_ALLOWED
- public static final int MQRC_SECURITY_ERROR
- public static final int MQRC_SEGMENT_LENGTH_ZERO
- public static final int MQRC_SELECTOR_COUNT_ERROR
- public static final int MQRC_SELECTOR_LIMIT_EXCEEDED
- public static final int MQRC_SELECTOR_ERROR

- public static final int MQRC_SELECTOR_NOT_FOR_TYPE
- public static final int MQRC_SERVICE_ERROR
- public static final int MQRC_SERVICE_NOT_AVAILABLE
- public static final int MQRC_SIGNAL_OUTSTANDING
- public static final int MQRC_SIGNAL_REQUEST_ACCEPTED
- public static final int MQRC_SIGNAL1_ERROR
- public static final int MQRC_SOURCE_BUFFER_ERROR
- public static final int MQRC_SOURCE_CCSID_ERROR
- public static final int MQRC_SOURCE_DECIMAL_ENC_ERROR
- public static final int MQRC_SOURCE_FLOAT_ENC_ERROR
- public static final int MQRC_SOURCE_INTEGER_ENC_ERROR
- public static final int MQRC_SOURCE_LENGTH_ERROR
- public static final int MQRC_SRC_ENV_ERROR
- public static final int MQRC_SRC_NAME_ERROR
- public static final int MQRC_STORAGE_CLASS_ERROR
- public static final int MQRC_STORAGE_NOT_AVAILABLE
- public static final int MQRC_SUPPRESSED_BY_EXIT
- public static final int MQRC_SYNCPOINT_LIMIT_REACHED
- public static final int MQRC_SYNCPOINT_NOT_AVAILABLE
- public static final int MQRC_TARGET_CCSID_ERROR
- public static final int MQRC_TARGET_DECIMAL_ENC_ERROR
- public static final int MQRC_TARGET_FLOAT_ENC_ERROR
- public static final int MQRC_TARGET_INTEGER_ENC_ERROR
- public static final int MQRC_TARGET_LENGTH_ERROR
- public static final int MQRC_TARGET_BUFFER_ERROR
- public static final int MQRC_TERMINATION_FAILED
- public static final int MQRC_TM_ERROR
- public static final int MQRC_TMC_ERROR
- public static final int MQRC_TRIGGER_CONTROL_ERROR
- public static final int MQRC_TRIGGER_DEPTH_ERROR
- public static final int MQRC_TRIGGER_MSG_PRIORITY_ERR
- public static final int MQRC_TRIGGER_TYPE_ERROR
- public static final int MQRC_TRUNCATED
- public static final int MQRC_TRUNCATED_MSG_ACCEPTED
- public static final int MQRC_TRUNCATED_MSG_FAILED
- public static final int MQRC_UNEXPECTED_ERROR
- public static final int MQRC_UNKNOWN_AUTH_ENTITY
- public static final int MQRC_UNKNOWN_DEF_XMIT_Q
- public static final int MQRC_UNKNOWN_ENTITY
- public static final int MQRC_UNKNOWN_Q_NAME

- public static final int MQRC_UNKNOWN_REF_OBJECT
- public static final int MQRC_UNKNOWN_XMIT_Q
- public static final int MQRC_UNKNOWN_ALIAS_BASE_Q
- public static final int MQRC_UNKNOWN_OBJECT_NAME
- public static final int MQRC_UNKNOWN_OBJECT_Q_MGR
- public static final int MQRC_UNKNOWN_REMOTE_Q_MGR
- public static final int MQRC_UNKNOWN_REPORT_OPTION
- public static final int MQRC_UOW_IN_PROGRESS
- public static final int MQRC_UOW_NOT_AVAILABLE
- public static final int MQRC_USER_ID_NOT_AVAILABLE
- public static final int MQRC_WAIT_INTERVAL_ERROR
- public static final int MQRC_WRONG_GMO_VERSION
- public static final int MQRC_WRONG_MD_VERSION
- public static final int MQRC_XMIT_Q_TYPE_ERROR
- public static final int MQRC_XMIT_Q_USAGE_ERROR
- public static final int MQRC_XQH_ERROR
- public static final int MQRC_XWAIT_CANCELED
- public static final int MQRC_XWAIT_ERROR

コンストラクタ

● public MQException(int completionCode, int reasonCode, Object source)

MQException コンストラクタです。

パラメタ

completionCode
　完了コードを指定します。

reasonCode
　理由コードを指定します。

source
　例外発生オブジェクトを指定します。

● public MQException(int completionCode, int reasonCode, Object source,String explanation)

MQException コンストラクタです。

パラメタ

completionCode
　完了コードを指定します。

reasonCode
　理由コードを指定します。

source
　例外発生オブジェクトを指定します。

explanation
　例外詳細情報を指定します。

MQGetMessageOptions クラス（Java）

このクラスは，MQQueue:get メソッドのオプションです。

変数

● public int options

MQQueue:get メソッドの動作を制御するオプションを指定します。

0 個以上の指定ができます。2 個以上必要とする場合，ビットの OR 演算子を使用して結合します。

MQGMO 構造体の，Options フィールドに相当します。

● public int waitInterval

ウェイト間隔を指定します。MQQueue:get メソッドで適合したメッセージの到着を待つ最大時間で，単位はミリ秒です。この時間が経過しても適合するメッセージが到着しなかった場合は，メソッドは完了コード MQC.MQCC_FAILED および理由コード MQException.MQRC_NO_MSG_AVAILABLE で例外が発生します。

この変数は options の MQC.MQGMO_WAIT と一緒に使用します。

このオプションが指定されている場合，waitInterval に 0 以上，または MQC.MQWI_UNLIMITED を指定してください。

MQGMO 構造体の，WaitInterval フィールドに相当します。

● public String resolvedQueueName

宛先キューの解決された名称です。

MQGMO 構造体の，ResolvedQName フィールドに相当します。

● public int matchOptions

検索するメッセージを判別する選択基準を指定します。

MQGMO 構造体の，MatchOptions フィールドに相当します。

● public char groupStatus

検索するメッセージがグループ内にあるかどうかを指示します。メッセージがグループ内にある場合は，グループ内の最後のメッセージかどうかを指示します。

MQGMO 構造体の，GroupStatus フィールドに相当します。

● public char segmentStatus

検索するメッセージが，論理メッセージのセグメントの一つであるかどうかを指示しま

す。メッセージが論理メッセージのセグメントの一つである場合，そのセグメントが最後のセグメントかどうかを指示します。

MQGMO 構造体の，SegmentStatus フィールドに相当します。

- public char segmentation

検索するメッセージが論理メッセージのセグメントの一つである場合，そのメッセージがセグメント化できるかどうかを指示します。

MQGMO 構造体の，Segmentation フィールドに相当します。

コンストラクタ

- public MQGetMessageOptions()

MQGetMessageOptions コンストラクタです。

options に MQC.MQGMO_NO_WAIT，waitInterval に 0，resolvedQueueName に空白を設定します。

MQManagedObject クラス（Java）

このクラスは，MQQueueManager，MQQueue，MQProcess のスーパクラスです。

変数

- public String alternateUserId

代替ユーザ識別子です。この変数は変更しないでください。

MQMD 構造体の，AlternateUserId フィールドに相当します。

- public String name

リソース名称です。この変数は変更しないでください。

- public int openOption

リソースをオープンしたオプションです。この変数は変更しないでください。

MQOPEN 命令の，Options 引数に相当します。

- public boolean isOpen

オープン済みフラグです。この変数は変更しないでください。

- public MQQueueManager connectionReference

キューマネジャ接続情報です。この変数は変更しないでください。

- public int closeOption

クローズオプションです。リソースをクローズするオプションを指定してください。

MQCLOSE 命令の，Options 引数に相当します。

コンストラクタ

- protected MQManagedObject()

MQManagedObject コンストラクタです。

メソッド

- public String getDescription() throws MQException

リソース記述子を取得します。

リソースの種類によって取得する内容が異なります。

- MQQueueManager：キューマネジャ記述子を取得します。
- MQQueue：キュー記述子を取得します。

- MQProcess：プロセス記述子を取得します。

● public void inquire(int selectors[], int intAttrs[],byte charAttrs[]) throws MQException

リソース属性を照会します。

MQINQ 命令に相当します。

パラメタ

selectors
　属性セレクタの配列を指定します。

intAttrs
　照会結果の整数型属性を格納する配列領域を指定します。

charAttrs
　照会結果の文字型属性を格納する領域を指定します。

● public Boolean isOpen()

オープン済みフラグです。このメソッドは変更しないでください。

● public synchronized void set(int selectors[], int intAttrs[],byte charAttrs[]) throws MQException

リソース属性を設定します。

MQSET 命令に相当します。

パラメタ

selectors
　属性セレクタの配列を指定します。

intAttrs
　整数型属性の配列を指定します。

charAttrs
　文字型属性を指定します。

● public synchronized void close() throws MQException

リソースをクローズします。

MQCLOSE 命令に相当します。

MQMessage クラス（Java）

このクラスは，MQ で登録・取り出しするメッセージのメッセージ記述子とメッセージバッファから形成されています。変数はメッセージ記述子の要素です。

メッセージ記述子の詳細については，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。

変数

- public int report

報告オプションです。

MQMD 構造体の，Report フィールドに相当します。

- public int messageType

メッセージタイプです。

MQMD 構造体の，MsgType フィールドに相当します。

- public int expiry

メッセージの生存期間です。

MQMD 構造体の，Expiry フィールドに相当します。

- public int feedback

フィードバックコードです。

MQMD 構造体の，Feedback フィールドに相当します。

- public int encoding

データのマシンコード化です。

MQMD 構造体の，Encoding フィールドに相当します。

- public int characterSet

コード化文字セット識別子です。

MQMD 構造体の，CodedCharSetId フィールドに相当します。

- public String format

フォーマット名称です。

MQMD 構造体の，Format フィールドに相当します。

● public int priority

メッセージ優先度です。

MQMD 構造体の，Priority フィールドに相当します。

● public int persistence

メッセージの永続性です。

MQMD 構造体の，Persistence フィールドに相当します。

● public byte messageId[]

メッセージ識別子です。

MQMD 構造体の，MsgId フィールドに相当します。

● public byte correlationId[]

相関識別子です。

MQMD 構造体の，CorrelId フィールドに相当します。

● public int backoutCount

バックアウトカウンタです。

MQMD 構造体の，BackoutCount フィールドに相当します。

● public String replyToQueueName

応答キューの名称です。

MQMD 構造体の，ReplyToQ フィールドに相当します。

● public String replyToQueueManagerName

応答キューマネジャの名称です。

MQMD 構造体の，ReplyToQMgr フィールドに相当します。

● public String userId

ユーザ識別子です。

MQMD 構造体の，UserIdentifier フィールドに相当します。

● public byte accountingToken[]

課金 token です。

MQMD 構造体の，AccountingToken フィールドに相当します。

● public String applicationIdData

アイデンティティに関連したアプリケーションデータです。

MQMD 構造体の，ApplIdentityData フィールドに相当します。

● public int putApplicationType

メッセージを登録したアプリケーションのタイプです。

MQMD 構造体の，PutApplType フィールドに相当します。

● public String putApplicationName

メッセージを登録したアプリケーションのアイデンティティです。

MQMD 構造体の，PutApplName フィールドに相当します。

● public GregorianCalender putDateTime

メッセージを登録した日時です。

MQMD 構造体の，PutDate,PutTime フィールドに相当します。

Java ではフォーマット形式は Date 型です。

● public String applicationOriginData

登録元に関連するアプリケーションデータです。

MQMD 構造体の，ApplOriginData フィールドに相当します。

● public byte[] groupId

物理メッセージが属しているメッセージグループを識別するバイト列です。

MQMD 構造体の，GroupId フィールドに相当します。

● public int messageSequenceNumber

グループ内の論理メッセージの順序番号です。

MQMD 構造体の，MsgSeqNumber フィールドに相当します。

● public int offset

セグメント化されたメッセージの場合，論理メッセージ先頭からの物理メッセージのデータのオフセットです。

MQMD 構造体の，Offset フィールドに相当します。

● public int messageFlags

メッセージのセグメント化と状況を制御するフラグです。

MQMD 構造体の，MsgFlags フィールドに相当します。

● public int originalLength

セグメント化されたメッセージの元の長さです。

MQMD 構造体の，OriginalLength フィールドに相当します。

コンストラクタ

● public MQMessage()

MQMessage コンストラクタです。

メッセージ記述子を初期化し，メッセージバッファを空にします。

メソッド

● public int getTotalMessageLength()

get したメッセージ長を取得します。

MQQueue:get したメッセージのバイト数です。MQQueue:get がメッセージ切り捨て例外を発生させた場合に，このメソッドでメッセージ長を取得できます。

● public int getMessageLength() throws IOException

get したメッセージバッファのバイト数を取得します。

● public int getDataLength() throws IOException

get したメッセージバッファの未読み込みデータバイト数を取得します。

● public void seek(int pos) throws IOException

get したメッセージバッファの現在読み込み位置を指定されたバイト数分移動します。

パラメタ

pos
　移動するバイト数を指定します。

● public void setDataOffset(int offset) throws IOException

get したメッセージバッファの読み込み位置を指定された位置に移動します。

パラメタ

offset
　バッファ先頭からのバイト数を指定します。

● public int getDataOffset() throws IOException

get したメッセージバッファの現在読み込み位置を取得します。

● public void clearMessage() throws IOException

put/get メッセージバッファをクリアします。

● public int getVersion()

メッセージ記述子バージョンを取得します。

● public void resizeBuffer(int size) throws IOException

get したメッセージバッファのサイズを変更します。

パラメタ

size
　変更するバイト数を指定します。

● public boolean readBoolean() throws IOException, EOFException

get したメッセージバッファの現在位置から boolean 型でデータを読み込みます。

● public char readChar() throws IOException, EOFException

get したメッセージバッファの現在位置から Unicode char 型でデータを読み込みます。

● public double readDouble() throws IOException, EOFException

get したメッセージバッファの現在位置から double 型でデータを読み込みます。

● public float readFloat() throws IOException, EOFException

get したメッセージバッファの現在位置から float 型でデータを読み込みます。

● public void readFully(byte b[]) throws IOException, EOFException

get したメッセージバッファの現在位置から指定されたバッファいっぱいにデータを読み込みます。

パラメタ

b
　データ取得バッファを指定します。

● public void readFully(byte b[], int off,int len) throws IOException, EOFException

get したメッセージバッファの現在位置から指定されたバッファの指定位置から指定サイズ分データを読み込みます。

パラメタ

b

データ取得バッファを指定します。

off

データ取得バッファ位置を指定します。

len

データ取得バイト数を指定します。

- public int readInt() throws IOException, EOFException

get したメッセージバッファの現在位置から int 型でデータを読み込みます。

- public int readInt4() throws IOException, EOFException

get したメッセージバッファの現在位置から int 型でデータを読み込みます。

- public String readLine() throws IOException

get したメッセージバッファの現在位置から終端文字（¥n，¥r，¥r¥n）までデータを読み込みます。

- public long readLong() throws IOException, EOFException

get したメッセージバッファの現在位置から long 型でデータを読み込みます。

- public long readInt8() throws IOException, EOFException

get したメッセージバッファの現在位置から long 型でデータを読み込みます。

- public short readShort() throws IOException, EOFException

get したメッセージバッファの現在位置から short 型でデータを読み込みます。

- public short readInt2() throws IOException, EOFException

get したメッセージバッファの現在位置から short 型でデータを読み込みます。

- public String readUTF() throws IOException

get したメッセージバッファの現在位置から UTF フォーマットでデータを読み込みます。

- public int readUnsignedByte() throws IOException, EOFException

get したメッセージバッファの現在位置から unsigned byte 型でデータを読み込みます。

- public int readUnsignedShort() throws IOException, EOFException

get したメッセージバッファの現在位置から unsigned short 型でデータを読み込みます。

● public int readUInt2() throws IOException, EOFException

get したメッセージバッファの現在位置から unsigned short 型でデータを読み込みます。

● public String readString(int length) throws IOException, EOFException

get したメッセージバッファの現在位置から指定バイト数分 String 型でデータを読み込みます。get したメッセージバッファは，characterSet メンバ変数で指定されるコード化文字セット識別子の String 型として，データ読み込み時に Unicode に変換されます。このメソッドでコード変換対象となるコード化文字セット識別子については，「付録 C MQMessage クラスで変換できるコード化文字セット識別子の一覧」を参照してください。

パラメタ

length
　読み込みバイト数を指定します。

● public void setVersion(int version)

メッセージ記述子バージョンを設定します。

パラメタ

version
　バージョンを指定します。

● public int skipBytes(int n) throws IOException, EOFException

get したメッセージバッファを指定バイト数分読み飛ばします。

パラメタ

n
　読み飛ばしバイト数を指定します。

● public void write(int b) throws IOException

put するメッセージバッファに 1 バイトを書き込みます。

パラメタ

b
　書き込みデータを指定します。

● public void write(byte b[]) throws IOException

put するメッセージバッファに指定バイト配列を書き込みます。

パラメタ

b
　書き込みデータバッファを指定します。

● public void write(byte b[], int off, int len) throws IOException

put するメッセージバッファに指定バイト配列の指定位置から指定バイト数分データを書き込みます。

パラメタ

b
　書き込みデータバッファを指定します。

off
　書き込み位置を指定します。

len
　書き込みバイト数を指定します。

● public void writeBoolean(boolean v) throws IOException

put するメッセージバッファに boolean 型でデータを書き込みます。

パラメタ

v
　書き込みデータを指定します。

● public void writeByte(int v) throws IOException

put するメッセージバッファに byte 型でデータを書き込みます。

パラメタ

v
　書き込みデータを指定します。

● public void writeBytes(String s) throws IOException

put するメッセージバッファに byte 型配列でデータを書き込みます。

パラメタ

s
　書き込みデータを指定します。

● public void writeChar(int v) throws IOException

put するメッセージバッファに Unicode char 型でデータを書き込みます。

パラメタ

v
　書き込みデータを指定します。

● public void writeChars(String s) throws IOException

put するメッセージバッファに Unicode char 型配列でデータを書き込みます。

パラメタ

s
　書き込みデータを指定します。

● public void writeDouble(double v) throws IOException

put するメッセージバッファに double 型でデータを書き込みます。

パラメタ

v
　書き込みデータを指定します。

● public void writeFloat(float v) throws IOException

put するメッセージバッファに float 型でデータを書き込みます。

パラメタ

v
　書き込みデータを指定します。

● public void writeInt(int v) throws IOException

put するメッセージバッファに int 型でデータを書き込みます。

パラメタ

v
　書き込みデータを指定します。

● public void writeInt4(int v) throws IOException

put するメッセージバッファに int 型でデータを書き込みます。

パラメタ

v
　書き込みデータを指定します。

● public void writeLong(long v) throws IOException

put するメッセージバッファに long 型でデータを書き込みます。

パラメタ

v
　書き込みデータを指定します。

● public void writeInt8(long v) throws IOException

put するメッセージバッファに long 型でデータを書き込みます。

パラメタ

v
　書き込みデータを指定します。

● public void writeShort(int v) throws IOException

put するメッセージバッファに short 型でデータを書き込みます。

パラメタ

v
　書き込みデータを指定します。

● public void writeInt2(int v) throws IOException

put するメッセージバッファに short 型でデータを書き込みます。

パラメタ

v
　書き込みデータを指定します。

● public void writeUTF(String str) throws IOException

put するメッセージバッファに UTF フォーマット型でデータを書き込みます。

パラメタ

str
　書き込みデータを指定します。

● public void writeString(String str) throws IOException

str パラメタで指定した String 型データを，characterSet メンバ変数で指定されるコード化文字セット識別子へ変換し，put するメッセージバッファに書き込みます。このメソッドでコード変換対象となるコード化文字セット識別子については，「付録 C MQMessage クラスで変換できるコード化文字セット識別子の一覧」を参照してください。

パラメタ

str

書き込みデータを指定します。

MQMessageTracker クラス（Java）

このクラスは，メッセージパラメタを，配布リスト内に指定された宛先用に調整します。

変数

- public int feedback

フィードバックコードです。

- public byte messageId[]

メッセージ ID です。

- public byte correlationId[]

相関識別子です。

- public byte accountingToken[]

課金トークンです。

- public byte[] groupId

グループ ID です。

MQProcess クラス（Java）

このクラスは，MQ のプロセス定義にアクセスします。

コンストラクタ

● public MQProcess(MQQueueManager qMgr, String processName, int openOptions, String queueManagerName, String alternateUserId) throws MQException

MQProcess コンストラクタです。

プロセス定義をオープンします。MQOPEN 命令での，MQOT_PROCESS の指定に相当します。

パラメタ

qMgr
　取得済み MQQueueManager オブジェクトを指定します。

processName
　プロセス定義名称を指定します。

openOptions
　オープンオプションを指定します。

queueManagerName
　キューマネジャ名称を指定します。

alternateUserId
　代替ユーザ識別子を指定します。

メソッド

● public String getApplicationId() throws MQException

アプリケーション識別子を取得します。

MQINQ 命令の，MQCA_APPL_ID セレクタに相当します。

● public int getApplicationType() throws MQException

アプリケーションタイプを取得します。

MQINQ 命令の，MQIA_APPL_TYPE セレクタに相当します。

● public String getEnvironmentData() throws MQException

環境データを取得します。

MQINQ 命令の，MQCA_ENV_DATA セレクタに相当します。

- public String getUserData() throws MQException

ユーザデータを取得します。

MQINQ 命令の，MQCA_USER_DATA セレクタに相当します。

MQPutMessageOptions クラス（Java）

このクラスは，MQQueue:put メソッドのオプションです。

変数

- public int options

MQQueue:put メソッドの動作を制御するオプションです。

0 個以上の指定ができます。2 個以上必要とする場合，ビットの OR 演算子を使用して結合します。

MQPMO 構造体の，Options フィールドに相当します。

- public MQQueue ContextReference

コンテキスト情報のソースを指示します。

MQPMO 構造体の，Context フィールドに相当します。

- public int recordFields

メッセージを配布リストに書き込むときに，キューごとにカスタマイズするフィールドを指示するフラグです。

MQPMO 構造体の，PutMsgRecFields フィールドに相当します。

- public String resolvedQueueName

宛先キューの解決された名称です。

MQPMO 構造体の，ResolvedQName フィールドに相当します。

- public String resolvedQueueManagerName

宛先キューの解決された名称です。

MQPMO 構造体の，ResolvedQMgrName フィールドに相当します。

- public int knownDestCount

正常に送信が完了したローカルキュー数です。

MQPMO 構造体の，knownDestCount フィールドに相当します。

- public int unknownDestCount

正常に送信が完了したリモートキュー数です。

MQPMO 構造体の，unknownDestCount フィールドに相当します。

● public int invalidDestCount

配布リスト内のキューに正常に送信できなかったメッセージ数です。

MQPMO 構造体の，invalidDestCount フィールドに相当します。

コンストラクタ

● public MQPutMessageOptions()

MQPutMessageOptions コンストラクタです。

options にオプションなし，resolvedQueueName および resolvedQueueManagerName に空白を設定します。

MQQueue クラス（Java）

このクラスは，MQ のキューにアクセスします。

コンストラクタ

● public MQQueue(MQQueueManager qMgr,String queueName,int openOptions, String queueManagerName,String dynamicQueueName,String alternateUserId) throws MQException

MQQueue コンストラクタです。

キューをオープンします。MQOPEN 命令での，MQOT_Q の指定に相当します。

パラメタ

qMgr
　取得済み MQQueueManager オブジェクトを指定します

queueName
　キュー名称を指定します。

openOptions
　オープンオプションを指定します。

queueManagerName
　キューマネジャ名称を指定します。

dynamicQueueName
　ダイナミックキュー名称を指定します。

alternateUserId
　代替ユーザ識別子を指定します。

メソッド

● public synchronized void get(MQMessage message, MQGetMessageOptions getMessageOptions, int MaxMsgSize) throws MQException

メッセージを取り出します。

MQGET 命令に相当します。

パラメタ

message
　MQMessage オブジェクトを指定します。

getMessageOptions
　MQGetMessageOptions オブジェクトを指定します。

MaxMsgSize
　最大メッセージ長を指定します。

● public synchronized void get(MQMessage message, MQGetMessageOptions getMessageOptions) throws MQException

メッセージを取り出します。メッセージサイズを取得するため，内部的に get を 2 回実行します。

MQGET 命令に相当します。

パラメタ

message
　MQMessage オブジェクトを指定します。

getMessageOptions
　MQGetMessageOptions オブジェクトを指定します。

● public synchronized void get(MQMessage message) throws MQException

メッセージを取り出します。メッセージサイズを取得するため，内部的に get を 2 回実行します。public synchronized void get(MQMessage message, MQGetMessageOptions getMessageOptions) throws MQException メソッドを単純化したものです。

MQGET 命令に相当します。

パラメタ

message
　MQMessage オブジェクトです。

● public synchronized void put(MQMessage message, MQPutMessageOptions putMessageOptions) throws MQException

メッセージを登録します。

MQPUT 命令に相当します。

パラメタ

message
　MQMessage オブジェクトです。

putMessageOptions
　MQPutMessageOptions オブジェクトです。

● public synchronized void put(MQMessage message) throws MQException

メッセージを登録します。public synchronized void put(MQMessage message, MQPutMessageOptions putMessageOptions) throws MQException メソッドを単純化したものです。

MQPUT 命令に相当します。

パラメタ

message
　MQMessage オブジェクトです。

● public GregorianCalender getCreationDateTime() throws MQException

キュー作成日時を取得します。

● public int getQueueType() throws MQException

次に示す値でキュータイプを取得します。

- MQC.MQQT_ALIAS：別名キューです。
- MQC.MQQT_LOCAL：ローカルキューです。
- MQC.MQQT_MODEL：モデルキューです。
- MQC.MQQT_REMOTE：リモートキューのローカル定義です。

MQINQ 命令の，MQIA_Q_TYPE セレクタに相当します。

● public int getCurrentDepth() throws MQException

カレントキューのメッセージ個数を取得します。

MQINQ 命令の，MQIA_CURRENT_Q_DEPTH セレクタに相当します。

● public int getDefinitionType() throws MQException

次に示す値でキュー定義タイプを取得します。

- MQC.MQQDT_PREDEFINED：事前定義キューです。
- MQC.MQQDT_PERMANENT_DYNAMIC：永続的動的キューです。
- MQC.MQQDT_TEMPORARY_DYNAMIC：一時的動的キューです。

MQINQ 命令の，MQIA_DEFINITION_TYPE セレクタに相当します。

● public int getMaximumDepth () throws MQException

キューの最大メッセージ個数を取得します。

MQINQ 命令の，MQIA_MAX_Q_DEPTH セレクタに相当します。

● public int getMaximumMessageLength() throws MQException

メッセージの最大長を取得します。

MQINQ 命令の，MQIA_MAX_MSG_LENGTH セレクタに相当します。

● public int getOpenInputCount() throws MQException

入力用にキューをオープンしている数を取得します。

MQINQ 命令の，MQIA_OPEN_INPUT_COUNT セレクタに相当します。

● public int getOpenOutputCount() throws MQException

出力用にキューをオープンしている数を取得します。

MQINQ 命令の，MQIA_OPEN_OUTPUT_COUNT セレクタに相当します。

● public int getShareability() throws MQException

次に示す値でキュー共用性を取得します。

- MQC.MQQA_SHAREABLE：共用できます。
- MQC.MQQA_NOT_SHAREABLE：共用できません。

MQINQ 命令の，MQIA_SHAREABILITY セレクタに相当します。

● public int getInhibitPut() throws MQException

次に示す値でメッセージ登録許可属性を取得します。

- MQC.MQQA_PUT_INHIBITED：登録禁止です。
- MQC.MQQA_PUT_ALLOWED：登録許可です。

MQINQ 命令の，MQIA_INHIBIT_PUT セレクタに相当します。

● public void setInhibitPut(int inhibit) throws MQException

メッセージ登録許可属性を設定します。

MQSET 命令の，MQIA_INHIBIT_PUT セレクタに相当します。

パラメタ

inhibit

登録許可属性です。次に示すどちらかの値を設定してください。

- MQC.MQQA_PUT_INHIBITED：登録を禁止します。
- MQC.MQQA_PUT_ALLOWED：登録を許可します。

● public int getInhibitGet() throws MQException

次に示す値でメッセージ取り出し許可属性を取得します。

- MQC.MQQA_GET_INHIBITED：取り出し禁止です。
- MQC.MQQA_GET_ALLOWED：取り出し許可です。

MQINQ 命令の，MQIA_INHIBIT_GET セレクタに相当します。

● public void setInhibitGet(int inhibit) throws MQException

メッセージ取り出し許可属性を設定します。

MQSET 命令の，MQIA_INHIBIT_GET セレクタに相当します。

パラメタ

inhibit

登録許可属性です。次に示すどちらかの値を設定してください。
- MQC.MQQA_GET_INHIBITED：取り出しを禁止します。
- MQC.MQQA_GET_ALLOWED：取り出しを許可します。

● public int getTriggerControl() throws MQException

次に示す値でトリガ制御情報を取得します。

- MQC.MQTC_OFF：トリガ制御をしません。
- MQC.MQTC_ON：トリガ制御をします。

MQINQ 命令の，MQIA_TRIGGER_CONTROL セレクタに相当します。

● public void setTriggerControl(int trigger) throws MQException

トリガ制御情報を設定します。

MQSET 命令の，MQIA_TRIGGER_CONTROL セレクタに相当します。

パラメタ

trigger

トリガ制御情報です。次に示すどちらかの値を設定してください。
- MQC.MQTC_OFF：トリガ制御をしません。
- MQC.MQTC_ON：トリガ制御をします。

● public String getTriggerData() throws MQException

トリガデータを取得します。

MQINQ 命令の，MQCA_TRIGGER_DATA セレクタに相当します。

● public void setTriggerData(String data) throws MQException

トリガデータを設定します。

MQSET 命令の，MQCA_TRIGGER_DATA セレクタに相当します。

パラメタ

data
　トリガデータ（最大長の MQC.MQ_TRIGGER_DATA_LENGTH）を指定します。

● public int getTriggerDepth() throws MQException

トリガのためのメッセージ登録数の下限値を取得します。

トリガタイプが MQC.MQTT_DEPTH の場合に，メッセージがキューにトリガのためのメッセージ登録数の下限値まで登録されたとき，トリガメッセージが発生します。

MQINQ 命令の，MQIA_TRIGGER_DEPTH セレクタに相当します。

● public void setTriggerDepth(int depth) throws MQException

トリガのためのメッセージ登録数の下限値を設定します。

トリガタイプが MQC.MQTT_DEPTH の場合に，メッセージがキューにトリガのためのメッセージ登録数の下限値まで登録されたとき，トリガメッセージが発生します。

MQSET 命令の，MQIA_TRIGGER_DEPTH セレクタに相当します。

パラメタ

depth
　トリガのためのメッセージ登録数の下限値を指定します。

● public int getTriggerMessagePriority() throws MQException

トリガに対するメッセージ優先度しきい値を取得します。

MQINQ 命令の，MQIA_TRIGGER_MSG_PRIORITY セレクタに相当します。

● public void setTriggerMessagePriority(int priority) throws MQException

トリガに対するメッセージ優先度しきい値を設定します。

MQSET 命令の，MQIA_TRIGGER_MSG_PRIORITY セレクタに相当します。

パラメタ

priority
　トリガに対するメッセージ優先度しきい値を指定します。

● public int getTriggerType() throws MQException

次に示す値でトリガタイプを取得します。

- MQC.MQTT_NONE：定義なしです。トリガは発生しません。
- MQC.MQTT_FIRST：キューが空き以外になった時に発生します。
- MQC.MQTT_EVERY：キューにメッセージが到着するごとに発生します。

- MQC.MQTT_DEPTH：キューに一定数のメッセージがある時に発生します。

MQINQ 命令の，MQIA_TRIGGER_TYPE セレクタに相当します。

● public void setTriggerType(int type) throws MQException

トリガタイプを設定します。

MQSET 命令の，MQIA_TRIGGER_TYPE セレクタに相当します。

パラメタ

type

トリガタイプです。次に示す値のどれかを設定してください。

- MQC.MQTT_NONE：定義なしです。トリガは発生しません。
- MQC.MQTT_FIRST：キューが空き以外になった時に発生します。
- MQC.MQTT_EVERY：キューにメッセージが到着するごとに発生します。
- MQC.MQTT_DEPTH：キューに一定数のメッセージがある時に発生します。

MQQueueManager クラス（Java）

このクラスは，MQ のキューマネジャにアクセスします。

変数

- public boolean isConnected

コネクション状態です。

コンストラクタ

- public MQQueueManager(String queueManagerName) throws MQException

MQQueueManager コンストラクタです。

コネクションを接続し，キューマネジャをオープンします。MQOPEN 命令の，MQOT_Q_MGR セレクタに相当します。

パラメタ

queueManagerName
　キューマネジャ名称を指定します。

メソッド

- public int getCharacterSet() throws MQException

コード化文字集合識別子を取得します。

MQINQ 命令の，MQIA_CODED_CHAR_SET_ID セレクタに相当します。

- public int getMaximumMessageLength() throws MQException

メッセージの最大長を取得します。

MQINQ 命令の，MQIA_MAX_MSG_LENGTH セレクタに相当します。

- public int getMaximumPriority() throws MQException

最大優先度を取得します。

MQINQ 命令の，MQIA_MAX_PRIORITY セレクタに相当します。

- public int getSyncpointAvailability() throws MQException

次に示す値で同期点の有効性を取得します。

- MQC.MQSP_AVAILABLE：コミット単位とシンクポイントが利用できます。
- MQC.MQSP_NOT_AVAILABLE：コミット単位とシンクポイントが利用できません。

MQINQ 命令の，MQIA_SYNCPOINT セレクタに相当します。

● public boolean getDistributionListCapable()

キューマネジャが配布リストをサポートするかどうかを指示します。

MQINQ 命令の，MQIA_DIST_LISTS セレクタに相当します。

● public synchronized void disconnect() throws MQException

キューマネジャへの接続を切り離します。

MQDISC 命令に相当します。

● public synchronized void commit() throws MQException

アプリケーションが同期点に達していることと，最後の同期点が永続化されてから発生した，すべてのメッセージの読み取りと書き込みをキューマネジャに指示します。

MQCMIT 命令に相当します。

● public synchronized void backout() throws MQException

最後の同期点以降に発生した，メッセージの読み取りと書き込みをすべてロールバックすることをキューマネジャに指示します。

MQBACK 命令に相当します。

● public synchronized MQQueue accessQueue(String queueName, int openOptions, String queueManagerName, String dynamicQueueName, String alternateUserId) throws MQException

キューをオープンします。

MQOPEN 命令の，MQOT_Q セレクタに相当します。

パラメタ

queueName
　キュー名称を指定します。

openOptions
　オープンオプションを指定します。

queueManagerName
　キューマネジャ名称を指定します。

dynamicQueueName
　ダイナミックキュー名称を指定します。

alternateUserId
　代替ユーザ識別子を指定します。

● public synchronized MQQueue accessQueue(String queueName, int openOptions) throws MQException

キューをオープンします。

MQOPEN 命令の，MQOT_Q セレクタに相当します。

パラメタ

queueName
　キュー名称を指定します。

openOptions
　オープンオプションを指定します。

● public synchronized MQProcess accessProcess(String processName, int openOptions, String queueManagerName, String alternateUserId) throws MQException

プロセス定義をオープンします。

MQOPEN 命令の，MQOT_PROCESS セレクタに相当します。

パラメタ

processName
　プロセス定義名称を指定します。

openOptions
　オープンオプションを指定します。

queueManagerName
　キューマネジャ名称を指定します。

alternateUserId
　代替ユーザ識別子を指定します。

● public synchronized MQProcess accessProcess(String processName, int openOptions) throws MQException

プロセス定義をオープンします。

MQOPEN 命令の，MQOT_PROCESS セレクタに相当します。

パラメタ

processName
　プロセス定義名称を指定します。

openOptions
　オープンオプションを指定します。

● public synchronized MQDistributionList accessDistributionList(MQDistributionListItem[] litems, int openOptions, String alternateUserId) throws MQException

配布リストをオープンします。

配布リスト用 MQOPEN 命令に相当します。

パラメタ

litems
　配布リストに組み込む項目を指定します。

openOptions
　オープンオプションを指定します。

alternateUserId
　代替ユーザ識別子を指定します。

● public synchronized MQDistributionList accessDistributionList(MQDistributionListItem[] litems, int openOptions)

配布リストをオープンします。

配布リスト用 MQOPEN 命令に相当します。

パラメタ

litems
　配布リストに組み込む項目を指定します。

openOptions
　オープンオプションを指定します。

● public synchronized void begin() throws MQException

新しい作業単位が開始されたことを示すシグナルをキューマネジャに送ります。

● public Boolean isConnected() throws MQException

isConnect の変数の値を戻します。

MQC インタフェース（Java）

TP1/Message Queue - Access で指定できる値，およびデフォルト値の定義の一覧を示します。

各変数の詳細については，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。

変数

- public final static int MQ_ACCOUNTING_TOKEN_LENGTH
- public final static int MQ_APPL_IDENTITY_DATA_LENGTH
- public final static int MQ_APPL_ORIGIN_DATA_LENGTH
- public final static int MQ_CHANNEL_NAME_LENGTH
- public final static int MQ_CLUSTER_NAME_LENGTH
- public final static int MQ_CORREL_ID_LENGTH
- public final static int MQ_CREATION_DATE_LENGTH
- public final static int MQ_CREATION_TIME_LENGTH
- public final static int MQ_FORMAT_LENGTH
- public final static int MQ_GROUP_ID_LENGTH
- public final static int MQ_MODE_NAME_LENGTH
- public final static int MQ_MSG_HEADER_LENGTH
- public final static int MQ_MSG_ID_LENGTH
- public final static int MQ_PROCESS_APPL_ID_LENGTH
- public final static int MQ_PROCESS_DESC_LENGTH
- public final static int MQ_PROCESS_ENV_DATA_LENGTH
- public final static int MQ_PROCESS_NAME_LENGTH
- public final static int MQ_PROCESS_USER_DATA_LENGTH
- public final static int MQ_PUT_TIME_LENGTH
- public final static int MQ_PUT_APPL_NAME_LENGTH
- public final static int MQ_PUT_DATE_LENGTH
- public final static int MQ_Q_DESC_LENGTH
- public final static int MQ_Q_MGR_DESC_LENGTH

- public final static int MQ_Q_MGR_NAME_LENGTH
- public final static int MQ_Q_NAME_LENGTH
- public final static int MQ_TRIGGER_DATA_LENGTH
- public final static int MQ_USER_ID_LENGTH
- public final static byte MQACT_NONE[]
- public final static int MQAT_UNKNOWN
- public final static int MQAT_NO_CONTEXT
- public final static int MQAT_CICS
- public final static int MQAT_MVS
- public final static int MQAT_IMS
- public final static int MQAT_OS2
- public final static int MQAT_DOS
- public final static int MQAT_AIX
- public final static int MQAT_UNIX
- public final static int MQAT_QMGR
- public final static int MQAT_OS400
- public final static int MQAT_WINDOWS
- public final static int MQAT_CICS_VSE
- public final static int MQAT_VMS
- public final static int MQAT_GUARDIAN
- public final static int MQAT_VOS
- public final static int MQAT_OPEN_TP1
- public final static int MQAT_XDM
- public final static int MQAT_TMS_4V
- public final static int MQAT_DEFAULT
- public final static int MQAT_USER_FIRST
- public final static int MQAT_USER_LAST
- public final static int MQAT_WINDOWS_NT
- public final static int MQCA_FIRST

- public final static int MQCA_APPL_ID
- public final static int MQCA_BASE_Q_NAME
- public final static int MQCA_CREATION_DATE
- public final static int MQCA_CREATION_TIME
- public final static int MQCA_DEAD_LETTER_Q_NAME
- public final static int MQCA_ENV_DATA
- public final static int MQCA_INITIATION_Q_NAME
- public final static int MQCA_PROCESS_DESC
- public final static int MQCA_PROCESS_NAME
- public final static int MQCA_Q_DESC
- public final static int MQCA_Q_MGR_DESC
- public final static int MQCA_Q_MGR_NAME
- public final static int MQCA_Q_NAME
- public final static int MQCA_REMOTE_Q_MGR_NAME
- public final static int MQCA_REMOTE_Q_NAME
- public final static int MQCA_NAMES
- public final static int MQCA_USER_DATA
- public final static int MQCA_TRIGGER_DATA
- public final static int MQCA_XMIT_Q_NAME
- public final static int MQCA_DEF_XMIT_Q_NAME
- public final static int MQCA_LAST_USED
- public final static int MQCA_LAST
- public final static int MQCA_CLUSTER_NAME
- public final static int MQCA_REPOSITORY_NAME
- public final static int MQCCSI_EMBEDDED
- public final static int MQCCSI_DEFAULT
- public final static int MQCCSI_INHERIT
- public final static int MQCCSI_Q_MGR
- public final static byte MQCI_NONE[]

- public final static int MQCO_NONE
- public final static int MQCO_DELETE
- public final static int MQCO_DELETE_PURGE
- public final static int MQDCC_NONE
- public final static int MQDL_NOT_SUPPORTED
- public final static int MQDL_SUPPORTED
- public final static int MQEI_UNLIMITED
- public final static int MQENC_NATIVE
- public final static int MQENC_INTEGER_MASK
- public final static int MQENC_DECIMAL_MASK
- public final static int MQENC_FLOAT_MASK
- public final static int MQENC_RESERVED_MASK
- public final static int MQENC_INTEGER_UNDEFINED
- public final static int MQENC_INTEGER_NORMAL
- public final static int MQENC_INTEGER_REVERSED
- public final static int MQENC_DECIMAL_UNDEFINED
- public final static int MQENC_DECIMAL_NORMAL
- public final static int MQENC_DECIMAL_REVERSED
- public final static int MQENC_FLOAT_UNDEFINED
- public final static int MQENC_FLOAT_IEEE_NORMAL
- public final static int MQENC_FLOAT_IEEE_REVERSED
- public final static int MQENC_FLOAT_S390
- public final static int MQFB_NONE
- public final static int MQFB_SYSTEM_FIRST
- public final static int MQFB_EXPIRATION
- public final static int MQFB_COA
- public final static int MQFB_COD
- public final static int MQFB_QUIT
- public final static int MQFB_APPL_CANNOT_BE_STARTED

- public final static int MQFB_TM_ERROR
- public final static int MQFB_APPL_TYPE_ERROR
- public final static int MQFB_STOPPED_BY_MSG_EXIT
- public final static int MQFB_XMIT_Q_MSG_ERROR
- public final static int MQFB_SYSTEM_LAST
- public final static int MQFB_APPL_FIRST
- public final static int MQFB_APPL_LAST
- public final static int MQFB_NOT_A_REPOSITORY_MSG
- public final static int MQFB_PAN
- public final static int MQFB_NAN
- public final static String MQFMT_NONE
- public final static String MQFMT_ADMIN
- public final static String MQFMT_DEAD_LETTER_HEADER
- public final static String MQFMT_EVENT
- public final static String MQFMT_PCF
- public final static String MQFMT_STRING
- public final static String MQFMT_TRIGGER
- public final static String MQFMT_XMIT_Q_HEADER
- public final static String MQFMT_DIST_HEADER
- public final static String MQFMT_MD_EXTENSION
- public final static String MQFMT_REF_MSG_HEADER
- public final static byte MQGI_NONE
- public final static int MQGMO_ALL_MSGS_AVAILABLE
- public final static int MQGMO_ALL_SEGMENTS_AVAILABLE
- public final static int MQGMO_COMPLETE_MSG
- public final static int MQGMO_WAIT
- public final static int MQGMO_NO_WAIT
- public final static int MQGMO_SYNCPOINT
- public final static int MQGMO_SYNCPOINT_IF_PERSISTENT

- public final static int MQGMO_NO_SYNCPOINT
- public final static int MQGMO_BROWSE_FIRST
- public final static int MQGMO_BROWSE_NEXT
- public final static int MQGMO_BROWSE_MSG_UNDER_CURSOR
- public final static int MQGMO_MSG_UNDER_CURSOR
- public final static int MQGMO_LOCK
- public final static int MQGMO_UNLOCK
- public final static int MQGMO_ACCEPT_TRUNCATED_MSG
- public final static int MQGMO_CURRENT_VERSION
- public final static int MQGMO_FAIL_IF_QUIESCING
- public final static int MQGMO_CONVERT
- public final static int MQGMO_NONE
- public final static int MQGMO_LOGICAL_ORDER
- public final static int MQGMO_VERSION_1
- public final static int MQGMO_VERSION_2
- public final static int MQGMO_VERSION_3
- public final static char MQGS_NOT_IN_GROUP
- public final static char MQGS_MSG_IN_GROUP
- public final static char MQGS_LAST_MSG_IN_GROUP
- public final static int MQIA_FIRST
- public final static int MQIA_APPL_TYPE
- public final static int MQIA_CODED_CHAR_SET_ID
- public final static int MQIA_CURRENT_Q_DEPTH
- public final static int MQIA_DEF_INPUT_OPEN_OPTION
- public final static int MQIA_DEF_PERSISTENCE
- public final static int MQIA_DEF_PRIORITY
- public final static int MQIA_DEFINITION_TYPE
- public final static int MQIA_HARDEN_GET_BACKOUT
- public final static int MQIA_INHIBIT_GET

- public final static int MQIA_INHIBIT_PUT
- public final static int MQIA_MAX_HANDLES
- public final static int MQIA_USAGE
- public final static int MQIA_MAX_MSG_LENGTH
- public final static int MQIA_MAX_PRIORITY
- public final static int MQIA_MAX_Q_DEPTH
- public final static int MQIA_MSG_DELIVERY_SEQUENCE
- public final static int MQIA_OPEN_INPUT_COUNT
- public final static int MQIA_OPEN_OUTPUT_COUNT
- public final static int MQIA_NAME_COUNT
- public final static int MQIA_Q_TYPE
- public final static int MQIA_RETENTION_INTERVAL
- public final static int MQIA_SHAREABILITY
- public final static int MQIA_TRIGGER_CONTROL
- public final static int MQIA_TRIGGER_INTERVAL
- public final static int MQIA_TRIGGER_MSG_PRIORITY
- public final static int MQIA_TRIGGER_TYPE
- public final static int MQIA_TRIGGER_DEPTH
- public final static int MQIA_SYNCPOINT
- public final static int MQIA_PLATFORM
- public final static int MQIA_MAX_UNCOMMITTED_MSGS
- public final static int MQIA_SCOPE
- public final static int MQIA_LAST_USED
- public final static int MQIA_LAST
- public final static int MQIA_DIST_LISTS
- public final static int MQIA_DEF_BIND
- public final static int MQIAV_NOT_APPLICABLE
- public final static int MQMD_VERSION_1
- public final static int MQMD_VERSION_2

- public final static int MQMD_CURRENT_VERSION
- public final static int MQMDS_PRIORITY
- public final static int MQMDS_FIFO
- public final static int MQMF_SEGMENTATION_INHIBITED
- public final static int MQMF_NONE
- public final static int MQMF_SEGMENTATION_ALLOWED
- public final static int MQMF_LAST_SEGMENT
- public final static int MQMF_SEGMENT
- public final static int MQMF_MSG_IN_GROUP
- public final static int MQMF_LAST_MSG_IN_GROUP
- public final static byte MQMI_NONE[]
- public final static int MQMO_NONE
- public final static int MQMO_MATCH_MSG_ID
- public final static int MQMO_MATCH_CORREL_ID
- public final static int MQMO_MATCH_GROUP_ID
- public final static int MQMO_MATCH_MSG_SEQ_NUMBER
- public final static int MQMO_MATCH_OFFSET
- public final static int MQMT_SYSTEM_FIRST
- public final static int MQMT_REQUEST
- public final static int MQMT_REPLY
- public final static int MQMT_DATAGRAM
- public final static int MQMT_REPORT
- public final static int MQMT_SYSTEM_LAST
- public final static int MQMT_APPL_FIRST
- public final static int MQMT_APPL_LAST
- public final static byte MQOII_NONE
- public final static int MQOL_UNDEFINED
- public final static int MQOO_INPUT_AS_Q_DEF
- public final static int MQOO_INPUT_SHARED

- public final static int MQOO_INPUT_EXCLUSIVE
- public final static int MQOO_BROWSE
- public final static int MQOO_OUTPUT
- public final static int MQOO_SAVE_ALL_CONTEXT
- public final static int MQOO_ALTERNATE_USER_AUTHORITY
- public final static int MQOO_FAIL_IF_QUIESCING
- public final static int MQOO_PASS_IDENTITY_CONTEXT
- public final static int MQOO_PASS_ALL_CONTEXT
- public final static int MQOO_SET_IDENTITY_CONTEXT
- public final static int MQOO_SET_ALL_CONTEXT
- public final static int MQOO_INQUIRE
- public final static int MQOO_SET
- public final static int MQOO_BIND_AS_Q_DEF
- public final static int MQOO_BIND_ON_OPEN
- public final static int MQOO_BIND_NOT_FIXED
- public final static int MQOT_Q
- public final static int MQOT_PROCESS
- public final static int MQOT_Q_MGR
- public final static int MQPER_PERSISTENT
- public final static int MQPER_NOT_PERSISTENT
- public final static int MQPER_PERSISTENCE_AS_Q_DEF
- public final static int MQPL_MVS
- public final static int MQPL_OS2
- public final static int MQPL_AIX
- public final static int MQPL_OS400
- public final static int MQPL_OPEN_TP1
- public final static int MQPL_XDM
- public final static int MQPL_TMS4V
- public final static int MQPMO_SYNCPOINT

- public final static int MQPMO_NO_SYNCPOINT
- public final static int MQPMO_NO_CONTEXT
- public final static int MQPMO_DEFAULT_CONTEXT
- public final static int MQPMO_PASS_IDENTITY_CONTEXT
- public final static int MQPMO_PASS_ALL_CONTEXT
- public final static int MQPMO_SET_IDENTITY_CONTEXT
- public final static int MQPMO_SET_ALL_CONTEXT
- public final static int MQPMO_ALTERNATE_USER_AUTHORITY
- public final static int MQPMO_FAIL_IF_QUIESCING
- public final static int MQPMO_NONE
- public final static int MQPMO_NEW_MSG_ID
- public final static int MQPMO_NEW_CORREL_ID
- public final static int MQPMO_LOGICAL_ORDER
- public final static int MQPMO_VERSION_1
- public final static int MQPMO_VERSION_2
- public final static int MQPMO_CURRENT_VERSION
- public final static int MQPMRF_NONE
- public final static int MQPMRF_MSG_ID
- public final static int MQPMRF_CORREL_ID
- public final static int MQPMRF_GROUP_ID
- public final static int MQPMRF_FEEDBACK
- public final static int MQPMRF_ACCOUNTING_TOKEN
- public final static int MQPRI_PRIORITY_AS_Q_DEF
- public final static int MQQA_GET_INHIBITED
- public final static int MQQA_GET_ALLOWED
- public final static int MQQA_PUT_INHIBITED
- public final static int MQQA_PUT_ALLOWED
- public final static int MQQA_SHAREABLE
- public final static int MQQA_NOT_SHAREABLE

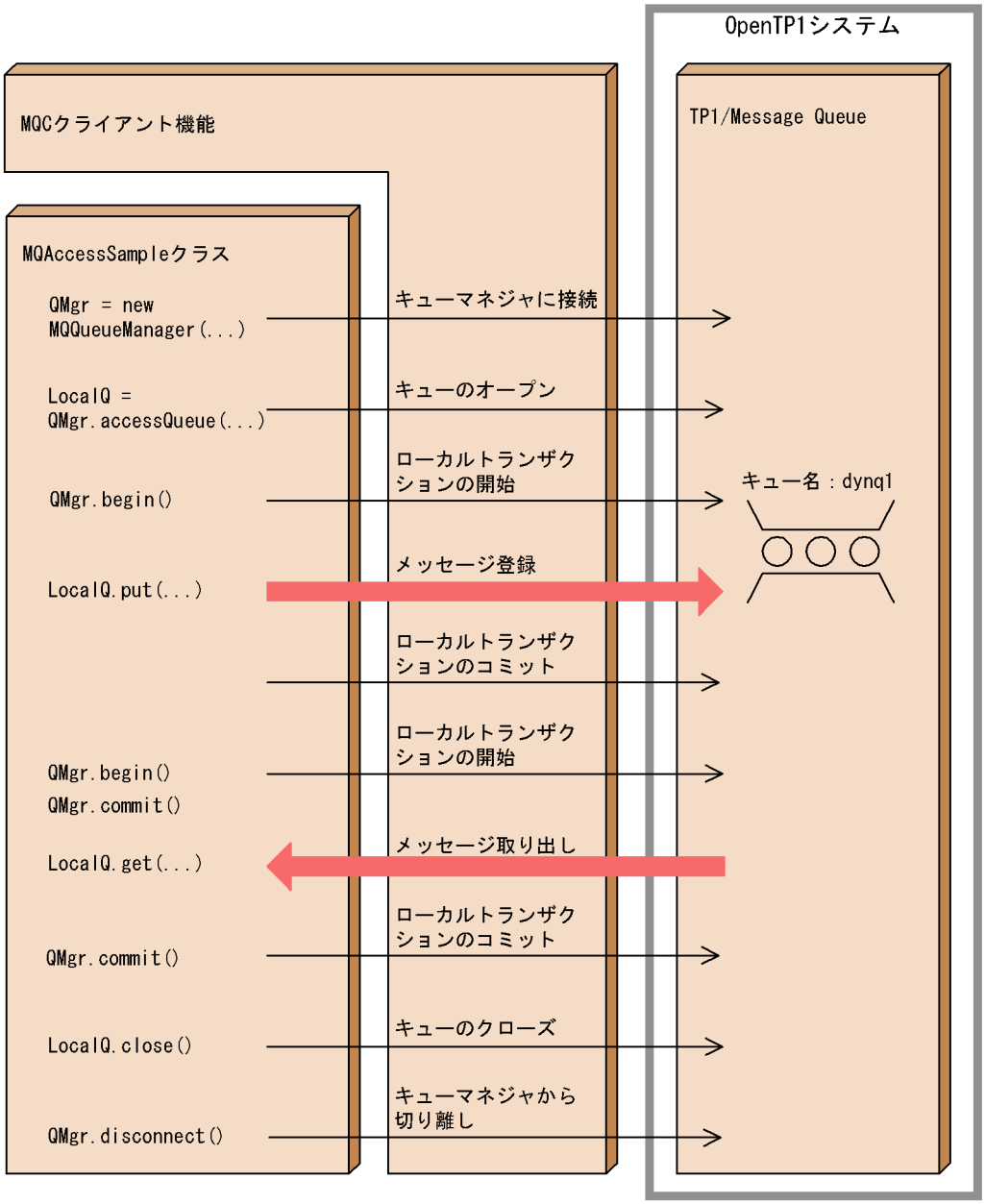
- public final static int MQQA_BACKOUT_HARDENED
- public final static int MQQA_BACKOUT_NOT_HARDENED
- public final static int MQQDT_PREDEFINED
- public final static int MQQDT_PERMANENT_DYNAMIC
- public final static int MQQDT_TEMPORARY_DYNAMIC
- public final static int MQQT_LOCAL
- public final static int MQQT_MODEL
- public final static int MQQT_ALIAS
- public final static int MQQT_REMOTE
- public final static int MQQT_CLUSTER
- public final static int MQRO_EXCEPTION
- public final static int MQRO_EXCEPTION_WITH_DATA
- public final static int MQRO_EXCEPTION_WITH_FULL_DATA
- public final static int MQRO_EXPIRATION
- public final static int MQRO_EXPIRATION_WITH_DATA
- public final static int MQRO_EXPIRATION_WITH_FULL_DATA
- public final static int MQRO_COA
- public final static int MQRO_COA_WITH_DATA
- public final static int MQRO_COA_WITH_FULL_DATA
- public final static int MQRO_COD
- public final static int MQRO_COD_WITH_DATA
- public final static int MQRO_COD_WITH_FULL_DATA
- public final static int MQRO_COPY_MSG_ID_TO_CORREL_ID
- public final static int MQRO_PASS_CORREL_ID
- public final static int MQRO_NEW_MSG_ID
- public final static int MQRO_PASS_MSG_ID
- public final static int MQRO_DEAD_LETTER_Q
- public final static int MQRO_DISCARD_MSG
- public final static int MQRO_NONE

- public final static int MQRO_REJECT_UNSUP_MASK
- public final static int MQRO_ACCEPT_UNSUP_MASK
- public final static int MQRO_ACCEPT_UNSUP_IF_XMIT_MASK
- public final static int MQRO_PAN
- public final static int MQRO_NAN
- public final static char MQSEG_INHIBITED
- public final static char MQSS_NOT_A_SEGMENT
- public final static int MQSP_AVAILABLE
- public final static int MQSP_NOT_AVAILABLE
- public final static char MQSS_LAST_SEGMENT
- public final static char MQSS_SEGMENT
- public final static int MQTC_OFF
- public final static int MQTC_ON
- public final static int MQTT_NONE
- public final static int MQTT_FIRST
- public final static int MQTT_EVERY
- public final static int MQTT_DEPTH
- public final static int MQUS_NORMAL
- public final static int MQUS_TRANSMISSION
- public final static int MQWI_UNLIMITED

Java のサンプルアプリケーション

Java のサンプルアプリケーションです。処理の流れを次の図に示します。

図 6-2 処理の流れ（Java の場合）



Java のサンプルコーディング

コーディング例を次に示します。

```
import JP.co.Hitachi.soft.MQ.Access.*;
                            // import the MQAccess package

public class MQAccessSample
{
  private String QMgrName = " ";  // queue manager name
  private MQQueueManager QMgr;    // MQQueueManager object
  public MQAccessSample()
  {
    try {
      // connect to the queue manager
      QMgr = new MQQueueManager(QMgrName);

      // set the open options
      int openOptions = MQC.MQOO_OUTPUT |
                                  MQC.MQOO_INPUT_AS_Q_DEF ;

      // open the queue
      MQQueue LocalQ =
          QMgr.accessQueue("dynq1", openOptions, null,
                                                 null, null);

      // create a message object
      MQMessage PutMessage = new MQMessage();
      // UTF format message data
  PutMessage.writeUTF("******** sample put data ********");

      // set the put message options
      MQPutMessageOptions mqpmo= new MQPutMessageOptions();
      mqpmo.options = MQC.MQPMO_SYNCPOINT;

      // begin local transaction
      QMgr.begin();

      // put the message
      LocalQ.put(PutMessage, mqpmo);

      // commit local transaction
      QMgr.commit();

      // create a message object
      MQMessage GetMessage = new MQMessage();

      // set the get message id
      GetMessage.messageId = PutMessage.messageId;

      // set the get message options
      MQGetMessageOptions mqgmo= new MQGetMessageOptions();
      mqgmo.options = MQC.MQGMO_SYNCPOINT |
                                       MQC.MQGMO_NO_WAIT;

      // begin local transaction
```

```
        QMgr.begin();

        // get the message
        LocalQ.get(GetMessage, mqgmo);

        // commit local transaction
        QMgr.commit();

        // display the get message
        String msgText = GetMessage.readUTF();
        System.out.println("The message is: " + msgText);

        // close the queue
        LocalQ.close();

        // disconnect from the queue manager
        QMgr.disconnect();
      }
      catch (MQException ex)
      {
        // MQ error
        System.out.println
        ("An MQ error occurred : Completion code " +
        ex.completionCode + " Reason code " + ex.reasonCode);
      }
      catch (java.io.IOException ex)
      {
        // buffer error
        System.out.println
          ("An error occurred whilst writing to the buffer:"
                                                    + ex);
      }
    }

    public static void main(String args[])
    {
      // execute this sample program
      MQAccessSample sample = new MQAccessSample();
    }
  }
```

7 MQC クライアント機能の JMS インタフェース

この章では，MQC クライアント機能の JMS インタフェースについて説明します。

JMS インタフェースの Java パッケージ

MQC クライアント機能の JMS インタフェース一覧

JMS インタフェース継承図

JMS インタフェースのメソッドと MQI の対応

JMS メッセージのヘッダとプロパティ

メッセージセレクタ

アプリケーション作成時の注意事項（JMS）

アプリケーション使用時の注意事項（JMS）

BytesMessage インタフェース（JMS）

ConnectionMetaData インタフェース（JMS）

DeliveryMode インタフェース（JMS）

Destination インタフェース・Queue インタフェース（JMS）

Message インタフェース（JMS）

MessageConsumer インタフェース・QueueReceiver インタフェース（JMS）

MessageProducer インタフェース・QueueSender インタフェース（JMS）

QueueBrowser インタフェース（JMS）

QueueConnection インタフェース（JMS）
QueueConnectionFactory インタフェース（JMS）
QueueSession インタフェース（JMS）
TemporaryQueue インタフェース（JMS）
Enumeration インタフェース（J2SE）
MQC インタフェース（JMS）
JMS インタフェースのサンプルアプリケーション
JMS インタフェースのサンプルコーディング

JMS インタフェースの Java パッケージ

MQC クライアント機能の JMS インタフェースは，パッケージとして次のインタフェースを提供します。

● jp.co.Hitachi.soft.mqadaptor
Java クラス（JMS インタフェース）パッケージです。

MQC クライアント機能の JMS インタフェース一覧

MQC クライアント機能が提供する JMS インタフェースの概要について説明します。

Java 言語の JMS インタフェースの一覧を次の表に示します。なお，Sun Microsystems, Inc. が提供する JMS 1.0 と MQC クライアント機能の JMS インタフェースとの機能差については，「付録 B　JMS 仕様と MQC クライアント機能の JMS インタフェースとの差異」を参照してください。

表 7-1　JMS インタフェースの一覧

クラス名	機能
BytesMessage	解釈されないバイトのストリームを含むメッセージです。
ConnectionMetaData	Connection を示すインフォメーションを提供します。
DeliveryMode	JMS によってサポートされる配送モードです。
Destination Queue	プロバイダ特有のキュー名をカプセル化します。
Message	すべての JMS メッセージのルートインタフェースです。
MessageConsumer QueueReceiver	クライアントはキューに届けられたメッセージを受け取ります。
MessageProducer QueueSender	クライアントはキューにメッセージを送ります。
QueueBrowser	クライアントがキューのメッセージを取り去らないで参照します。
QueueConnection Connection	JMS PTP プロバイダへのアクティブな接続です。
QueueConnectionFactory ConnectionFactory	クライアントが JMS PTP プロバイダで QueueConnection を生成するために QueueConnectionFactory を使用します。
QueueSession Session	QueueReceiver，QueueSender，QueueBrowser，および TemporaryQueue を作るためのメソッドを提供します。
TemporaryQueue	QueueConnection がアクティブ状態のときに生成されるユニークな Queue オブジェクトです。

インタフェースの一覧を次の表に示します。

表 7-2　インタフェースの一覧（JMS インタフェース）

インタフェース名	機能
MQC	TP1/Message Queue の定義値です。

JMS インタフェース使用時の Java 例外クラスの一覧を次の表に示します。

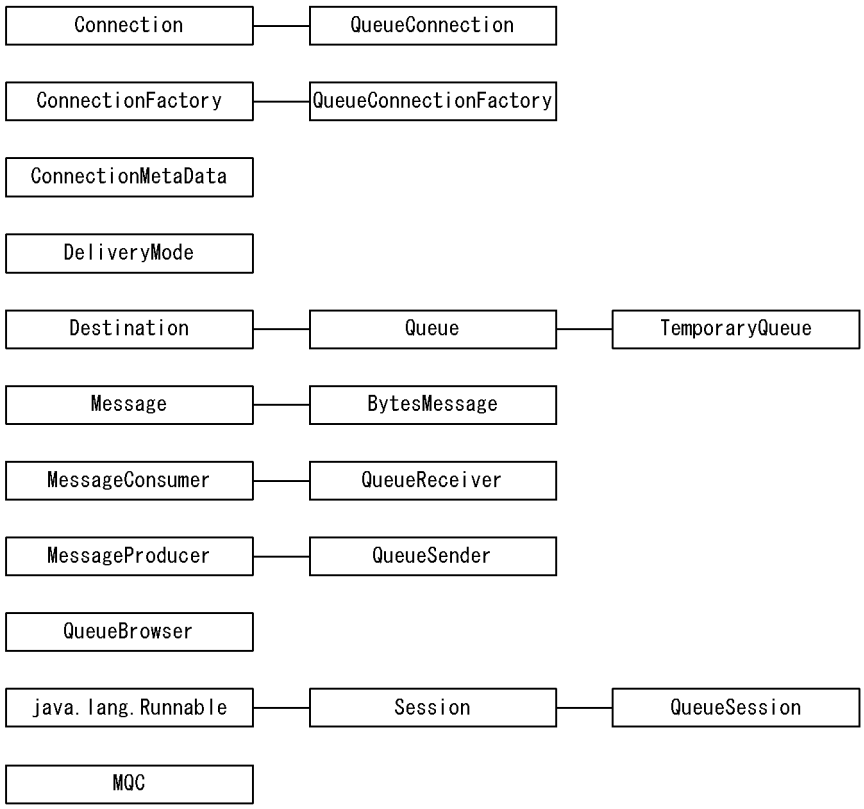
表 7-3 例外クラスの一覧（JMS インタフェース）

クラス名	例外をスローするタイミング
IllegalStateException	メソッドが不正な場合，不適当なタイミングで起動された場合，またはプロバイダが要求されたオペレーションを実行するための適切な状態でない場合
InvalidClientIDException	クライアントが接続のクライアント ID をプロバイダによって拒否される値にセットしようとした場合
InvalidDestinationException	宛先がプロバイダによって理解されない，つまり宛先が有効でない場合
InvalidSelectorException	JMS クライアントが不正な構文のメッセージセレクタをプロバイダに指定しようとした場合
JMSException	すべての JMS 例外のルートクラスであるため，特定のタイミングに該当しません。
JMSSecurityException	クライアントによって送られたユーザ名またはパスワードが，プロバイダによって拒否された場合
MessageEOFException	StreamMessage か BytesMessage が読みこまれている最中に，ストリームが予期しないで終了に達した場合
MessageFormatException	JMS クライアントがメッセージによってサポートされていないデータタイプを使おうとした場合，またはメッセージデータを間違ったタイプで読み込もうとした場合
MessageNotReadableException	JMS クライアントが書き込み専用のメッセージを読み込もうとした場合
MessageNotWriteableException	JMS クライアントが読み込み専用のメッセージに書き込もうとした場合
ResourceAllocationException	プロバイダがメソッドによって要求されたリソースを割り当てられない場合
TransactionInProgressException	トランザクションが進行中であるために，オペレーションが無効の場合
TransactionRolledBackException	Session.commit の呼び出しの結果が現在のトランザクションのロールバックに帰着する場合

JMS インタフェース継承図

JMS インタフェースの継承図と例外クラスの継承図を次の図に示します。

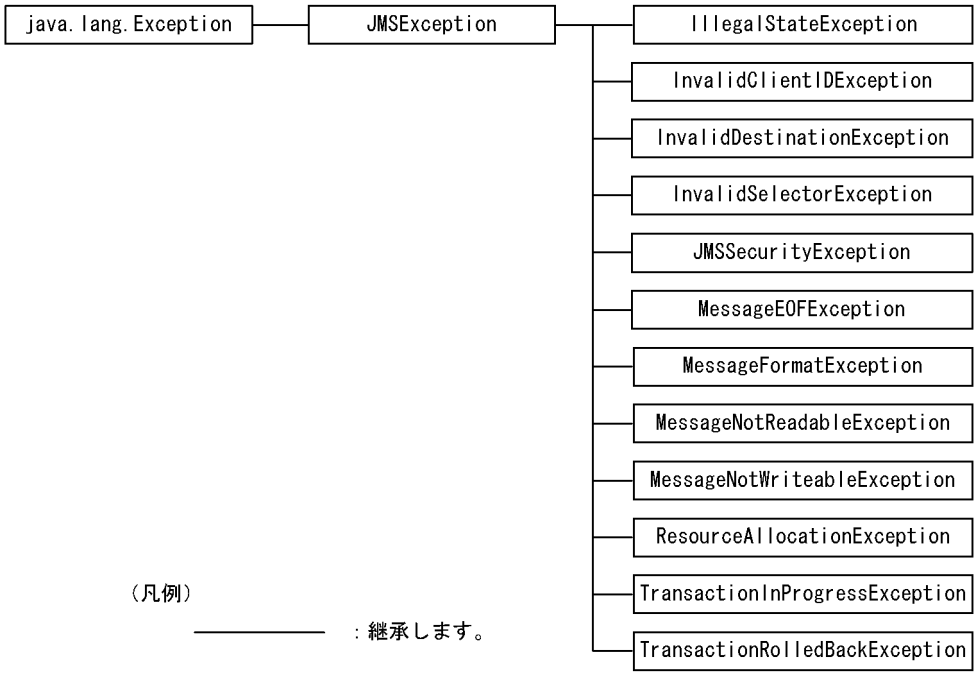
図 7-1 JMS インタフェース継承図



（凡例）

――――― ：継承します。

図 7-2 例外クラス継承図



JMS インタフェースのメソッドと MQI の対応

JMS インタフェースのメソッドの発行時に，内部で MQI が発行される場合があります。その対応を次の表に示します。

表 7-4　メソッド発行時に内部で発行される MQI

Queue オブジェクト使用時		TemporaryQueue オブジェクト使用時		発行される MQI
インタフェース	メソッド	インタフェース	メソッド	
QueueSession	createSender createReceiver	QueueSession	createTemporary Queue	MQOPEN
QueueBrowser	getEnumeration			
QueueSender QueueReceiver QueueBrowser	close	QueueSession	close	MQCLOSE
QueueSender	send （Queue 未指定時）	QueueSender	send （Queue 未指定時）	MQPUT
QueueSender	send （Queue 指定時）	QueueSender	send （Queue 指定時）	MQPUT1
QueueReceiver	receive receiveNoWait	QueueReceiver	receive receiveNoWait	MQGET
Enumeration ※	hasMoreElements nextElement	Enumeration ※	hasMoreElements nextElement	

注※

QueueBrowser インタフェースの getEnumeration メソッドによって取得した Enumeration オブジェクトです。

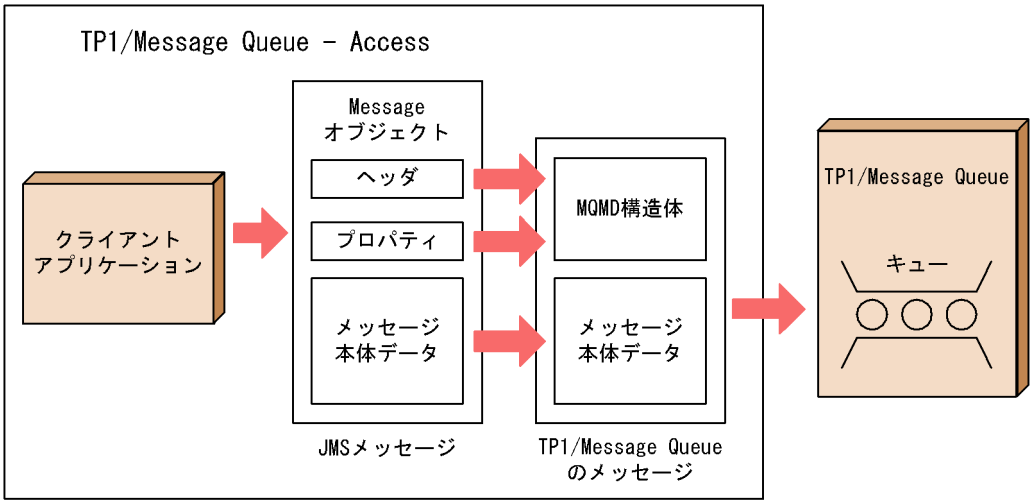
なお，Cosminexus Component Container のアソシエーション機能によって，各キューに対して MQCLOSE，MQOPEN の順に発行されます。

JMS メッセージのヘッダとプロパティ

JMS メッセージと TP1/Message Queue のメッセージとの対応

MQC クライアント機能の JMS インタフェースで作成したメッセージ（JMS メッセージ）と，TP1/Message Queue のメッセージとの対応を次の図に示します。

図 7-3　JMS メッセージと TP1/Message Queue のメッセージとの対応



JMS メッセージは，ヘッダ，プロパティ，およびメッセージ本体で構成されています。

一般的な JMS メッセージのヘッダは，クライアントとプロバイダの双方がメッセージを識別・送信するための値を含みます。プロパティは，アプリケーション固有の値を格納するためのフィールドを定義する組み込み機能を提供します。

TP1/Message Queue - Access では，図 7-3 に示すとおり，JMS メッセージのヘッダとプロパティが MQMD 構造体に対応づけられてキューに格納されます。また，プロパティは，MQMD 構造体の各フィールドに値を設定・取得するために使用されます。アプリケーション固有のプロパティ値は TP1/Message Queue のキューには格納されません。

JMS ヘッダと対応する MQMD 構造体のフィールド，および JMS プロパティと対応する MQMD 構造体のフィールドを次の表に示します。設定者が「クライアント」以外のヘッダは指定できません。

表 7-5 JMS ヘッダと対応する MQMD 構造体のフィールド

ヘッダ	型	対応する MQMD 構造体のフィールド	設定者	備考
JMSDestination	Destination	－	送信時プロバイダ	送信時に設定しても無視されます。
JMSMessageID	String	MsgId	送信時プロバイダ	送信時に設定しても無視されます。
JMSTimestamp	long	PutDate, PutTime	送信時プロバイダ	送信時に設定しても無視されます。
JMSRedelivered	boolean	BackoutCount	受信時プロバイダ	送信時に設定しても無視されます。
JMSCorrelationID	String ※1	CorrelId	クライアント（Message）	－
JMSReplyTo	Destination	ReplyToQ, ReplyToQMgr	クライアント（Message）	－
JMSType	String	－	クライアント（Message）	TP1/Message Queue では使用されません。
JMSDeliveryMode	int	Persistence	クライアント（QueueSender）	－
JMSExpiration	long	Expiry	クライアント（QueueSender）	－
JMSPriority	int	Priority	クライアント（QueueSender）	－

表 7-6 JMS プロパティと対応する MQMD 構造体のフィールド

プロパティ	型	対応する MQMD 構造体のフィールド	設定者	備考
JMSXAppID	String	PutApplName	送信時プロバイダ	送信時に設定しても無視されます。
JMSXUserID	String	UserIdentifier	送信時プロバイダ	送信時に設定しても無視されます。
JMSXDeliveryCount	int	BackoutCount	受信時プロバイダ	送信時に設定しても無視されます。
JMSXGroupID ※1	String ※2	GroupId	クライアント（Message）	－
JMSXGroupSeq	int	MsgSeqNumber	クライアント（Message）	－
JMS_HITACHI_PutApplType	int	PutApplType	送信時プロバイダ	送信時に設定しても無視されます。

プロパティ	型	対応する MQMD 構造体のフィールド	設定者	備考
JMS_HITACHI_CodedCharSetId	int	CodedCharSetId	クライアント（Message）	－
JMS_HITACHI_Encoding	int	Encoding	クライアント（Message）	－
JMS_HITACHI_Feedback	int	Feedback	クライアント（Message）	－
JMS_HITACHI_MsgType	int	MsgType	クライアント（Message）	－
JMS_HITACHI_Report	int	Report	クライアント（Message）	－
JMS_HITACHI_Format	String※3	Format	クライアント（Message）	－
JMS_HITACHI_MQMF_LAST_MSG_IN_GROUP※1	boolean	MsgFlags	クライアント（Message）	－

（凡例）

－：該当しません。

注※ 1

プロパティ JMSXGroupID の文字列が有効かどうか，またプロパティ JMS_HITACHI_MQMF_LAST_MSG_IN_GROUP の値によって，MQMD 構造体の MsgFlags フィールドに次の値が代入されます。

JMSXGroupID の文字列	JMS_HITACHI_MQMF_LAST_MSG_IN_GROUP の値	
	false	true
無効	MQMF_NONE	
有効	MQMF_MSG_IN_GROUP	MQMF_LAST_MSG_IN_GROUP

注※ 2

String 型から長さ 24 バイトの byte 型配列（TP1/Message Queue の MQBYTE24 型）に変換された結果，24 バイトを超える部分は切り捨てられます。

文字列の先頭が「ID:」ではない場合，UTF-8 ストリングとしてエンコードされます。さらに，メッセージの送信を契機に，ヘッダとプロパティに格納された文字列は，先頭が「ID:」で以降が 48 けたの 16 進数の文字列に置き換えられます。

文字列の先頭が「ID:」の場合，「ID: ＋ 16 進数」と解釈されて 16 進数が MQBYTE24 型に変換されます。この場合，16 進数の部分が 48 けたに満たないときは，不足部分は 48 けたまで「0」を補われて MQBYTE24 型に変換されます。ま

た，16 進数の部分が 48 けたを超えるときは，超える部分は切り捨てられます。
次に示す 1. ～ 4. の文字列にあてはまる場合は無効な文字列と見なされ，対応する MQMD 構造体のフィールドには MQCI_NONE または MQGI_NONE が格納されます。

1. null の場合
2. 先頭が「ID:」で，後続の文字がない場合
3. 先頭が「ID:」で，後続の文字は 48 文字目まですべて「0」の場合
4. 先頭が「ID:」で，後続の文字が 16 進数として解釈できない場合

注※ 3

String 型から長さ 8 バイトの char 型配列（TP1/Message Queue の MQCHAR8 型）に変換（UTF-8 から 7 ビット ASCII に変換）された結果，8 バイトを超える部分は切り捨てられます。

JMS プロパティ読み込み時の型変換

JMS プロパティに設定できる型は，boolean，byte，short，int，long，float，double，および String です。書き込まれたときとは異なる型で JMS プロパティを読み込む場合，次の表に示すように型が変換されます。

表 7-7 JMS プロパティの型変換

プロパティの型		読み込むときの型							
		boolean	byte	short	int	long	float	double	String
書き込まれたときの型	boolean	○							○
	byte		○	○	○	○			○
	short			○	○	○			○
	int				○	○			○
	long					○			○
	float						○	○	○
	double							○	○
	String ※	○	○	○	○	○	○	○	○

（凡例）

○：型が変換されます。
空白：型が変換されません。この場合，読み込み時に例外がスローされます。

注※

String 型で書き込まれた文字列が変換される場合，書き込まれた文字列が変換後の型のラッパークラスの valueOf メソッドによって解釈ができないときは，例外がス

ローされる場合があります。

メッセージセレクタ

QueueSession.createReceiver メソッドまたは createBrowser メソッドの引数に，メッセージセレクタを指定できます。メッセージセレクタは，受信するメッセージの条件を指定するための java.lang.String 型の構文です。受信側アプリケーションはメッセージセレクタを使用することによって，構文の条件に合うヘッダの値を持つメッセージをキューから受信できます。

メッセージセレクタには，識別子，比較演算子，リテラル，および論理演算子を組み合わせた構文を指定します。

識別子

次に示す識別子を指定できます。

- ヘッダの JMSMessageID
- ヘッダの JMSCorrelationID

識別子は半角英字であり，大文字と小文字が区別されます。識別子はそれぞれ 1 回だけ指定できます。不正な識別子を指定した場合，または同じ識別子を複数回指定した場合は，InvalidSelectorException が発生します。

比較演算子

等号（=）だけを指定できます。比較演算子は半角記号です。不正な比較演算子を指定した場合は，InvalidSelectorException が発生します。

リテラル

各識別子に指定できるリテラルを次に示します。不正なリテラルを指定した場合は，InvalidSelectorException が発生します。

- ヘッダの JMSMessageID または JMSCorrelationID の場合
  文字列リテラルを指定できます。
  文字列リテラルはアポストロフィ（'）で囲んでください。文字列リテラルにアポストロフィ（'）を含む場合はアポストロフィ（'）を二つ記述します。例えば "literal's" を指定する場合は，次に示すとおり記述します。

  ```
  'literal''s'
  ```

  なお，リテラルは次のどちらかの形式で指定できます。指定されたリテラルは受信時に，JMSMessageID ヘッダは MQMD の MsgId フィールドに，JMSCorrelationID ヘッダは MQMD の CorrelId フィールドに変換されます。
- 文字列の先頭が「ID:」ではない場合，UTF-8 ストリングとしてエンコードされます。String 型から長さ 24 バイトの byte 型配列（TP1/Message Queue の MQBYTE24 型）に変換された結果，24 バイトを超える部分は切り捨てられます。

- 文字列の先頭が「ID:」の場合，「ID: ＋ 16 進数」と解釈されて 16 進数が MQBYTE24 型に変換されます。この場合，16 進数の部分が 48 けたに満たないときは，不足部分は 48 けたまで「0」を補われて MQBYTE24 型に変換されます。また，16 進数の部分が 48 けたを超えるときは，超える部分は切り捨てられます。
  次に示す 1. ～ 4. の文字列にあてはまる場合は無効な文字列と見なされ，対応する MQMD 構造体のフィールドには MQMI_NONE，または MQCI_NONE が格納されます。
  1. null の場合
  2. 先頭が「ID:」で，後続の文字がない場合
  3. 先頭が「ID:」で，後続の文字は 48 文字目まですべて「0」の場合
  4. 先頭が「ID:」で，後続の文字が 16 進数として解釈できない場合

論理演算子

AND だけを指定できます。論理演算子は半角英字であり，大文字だけを指定できます。論理演算子の前後には空白が必要です。

上記条件を満たさない場合は，InvalidSelectorException が発生します。

指定できる構文の例

メッセージセレクタに指定できる構文の例を次に示します。

```
JMSMessageID = 'aaa'
JMSCorrelationID = 'bbb'
JMSMessageID = 'aaa' AND JMSCorrelationID = 'bbb'
JMSCorrelationID = 'aaa' AND JMSMessageID = 'bbb'
```

アプリケーション作成時の注意事項（JMS）

- TP1/Message Queue - Access の JMS インタフェースでは，論理コネクション（QueueSession）と物理コネクションを 1 対 1 で対応づける必要があります。このため，アプリケーション（EJB）の Resource Reference タブで必ず Unshareable を設定してください。Shareable を設定して，アプリケーションで複数の QueueSession を作成すると，例外がスローされます。
- QueueSession，および QueueSession を使用して生成したオブジェクトを複数のスレッドやアプリケーション間で共有することはできません。複数のスレッドやアプリケーション間で共有した場合の動作は保証できません。
- 一つの QueueSession が生成されるたびに，一つの物理コネクションが消費されます。このため，アプリケーションで使用しなくなった QueueSession に対しては必ず QueueSession の close メソッドを発行し，物理コネクションを解放するようにしてください。
- QueueSession の生成時に transacted パラメタに true を指定した場合の注意事項を次に示します。
  - 該当する QueueSession から生成した QueueSender，QueueReceiver オブジェクトからの send または receive メソッドの発行を契機に，ローカルトランザクションが開始されます。したがって，これらのメソッドの発行前に QueueSession の commit または rollback メソッドを発行すると，例外がスローされます。
  - 該当する QueueSession から生成した QueueSender，QueueReceiver オブジェクトからの send または receive メソッドの発行前にユーザトランザクションを開始したときは，send または receive メソッドはユーザトランザクションに含まれます。ユーザトランザクションによってトランザクションを開始した場合は，ユーザトランザクションから commit または rollback メソッドを発行してください。ユーザトランザクションが開始済みの状態で QueueSession から commit または rollback メソッドを発行すると，例外がスローされます。
  - コミットまたはロールバックをするためには，QueueSession オブジェクトの commit または rollback メソッドを使用してください。
    なお，ローカルトランザクションが開始済みの状態のときは，ユーザトランザクションのコミットまたはロールバックは受け付けられません。
- QueueConnection の createQueueSession メソッドと QueueSession の close メソッドを繰り返し発行すると，QueueConnection の createQueueSession メソッドで回線障害によってコネクション確立に失敗し，例外がスローされる場合があります。このため，このようなアプリケーションを作成する場合はコネクションプーリング機能を使用してください。
- QueueReceiver の生成時に指定するキューが共用できない属性（Shareability 属性が MQQA_NOT_SHAREABLE）の場合，QueueReceiver を複数生成できません。

QueueReceiver を複数生成する場合は，共用できる属性（Shareability 属性が MQQA_SHAREABLE）のキューを指定して，QueueReceiver を生成するようにしてください。

- 次に示すメソッドを発行してキューからメッセージを取得する場合は，メッセージデータはコード変換されません。
  - QueueReceiver インタフェースの receive メソッドおよび receiveNoWait メソッド
  - QueueBrowser インタフェースの getEnumeration メソッドによって取得した Enumeration オブジェクトの nextElement メソッド
- Message-Driven Bean アプリケーションを使用した場合，アプリケーションの Properties の Pooled Instances で Maximum に指定した値の分だけ Message-Driven Bean キュー監視機能の監視スレッドが起動してキューを監視します。Maximum に 0 を指定した場合は，監視スレッド数は 1 となります。
- Message-Driven Bean を使用する場合，リソースアダプタのトランザクションサポートのレベルを XATransaction に指定してください。
  XATransaction 以外を指定すると，Message-Driven Bean を含むアプリケーションのデプロイに失敗します。

アプリケーション使用時の注意事項（JMS）

- リソースアダプタのトランザクションサポートのレベルに XATransaction を指定する場合，Cosminexus Component Container のライトトランザクションの設定を無効にする必要があります。
  ライトトランザクションの設定を無効にする場合は，"/opt/Cosminexus/CC/server /usrconf/ejb/< サーバ名 >/usrconf.properties" ファイルに次の指定をしてください。

  ```
  ejbserver.distributedtx.XATransaction.enabled=true
  ```

  false を指定した場合，または指定を省略した場合は，リソースアダプタのデプロイに失敗します。

- MDB キュー監視機能で監視するキューを二つ以上のスレッドで監視する場合，サーバ側で作成するキューの属性を同時に何度も入力用にオープンできるようにする必要があります。mqaqueatl（モデルキューの属性定義）の -s オプション（共用性）に shareble（キューを共用できます）を指定してください。notshareble（キューを共用できません）を指定した場合，起動する監視スレッドは一つです。

- JMS インタフェースでの物理コネクションの使用数では，アプリケーションによる QueueSession の同時使用数以外に一つ以上の物理コネクションを Cosminexus Component Container が使用する場合があります。このため，MQC ゲートウェイサーバはアプリケーションによる QueueSession の同時使用数より多く起動しておく必要があります。Cosminexus Component Container が使用する物理コネクションの数は Cosminexus Component Container の「リリースノート」を参照してください。

- MQC リスナサーバ通信環境定義の要求受信監視タイマ値はクライアントアプリケーションが QueueSession を生成してからクローズするまでの間，切断されることのないように設定する必要があります。
  QueueSession のクローズ前に MQC リスナサーバからコネクションが切断されると QueueSession および QueueSession を使用して生成したオブジェクトに対して発行したメソッドが例外を返すことがあります。

- Cosminexus Component Container のコネクションプーリング機能を使用する場合，MQC リスナサーバ通信環境定義の要求受信監視タイマ値は Cosminexus Component Container のコネクションタイムアウト値より長く設定する必要があります。MQC リスナサーバ通信環境定義の要求受信監視タイマ値が Cosminexus Component Container のコネクションタイムアウト値より短いと MQC リスナサーバで要求受信監視によるタイムアウトが発生する場合があります。なお，タイムアウトとなったコネクションはコネクションプーリング機能によって再利用されるときに，再確立されます。

- Cosminexus Component Container のコネクションプーリング機能を使用するかどうかに関係なく，ユーザアプリケーションが QueueSession を Cosminexus Component

Container のユーザトランザクション中の状態でクローズをするとその物理コネクションは切断されないでトランザクション決着待ちの状態で継続されます。このため，このようなアプリケーションを作成する場合は，MQC ゲートウェイサーバをコネクションの継続分を考慮して起動しておく必要があります。なお，トランザクション決着待ち状態で継続したコネクションはアプリケーションからユーザトランザクションのコミット，ロールバック，アプリケーションの停止，または Cosminexus Component Container のトランザクションタイムアウトの検知によって切断されます。

- MDB キュー監視機能の監視スレッドが異常停止すると，「Message-driven Bean:<Bean 名 > caught an error from TP1 Message Queue. The error message is :」のあとに KFCA31345-W メッセージがコンソールに出力されます。MDB アプリケーションをアンデプロイする前にすべての監視スレッドが停止すると，KFCA31346-E のメッセージが出力されます。すべての監視スレッドが停止した場合，一度 MDB アプリケーションをアンデプロイし，再度 MDB アプリケーションをデプロイしてください。監視スレッドの異常停止に関する対策についての詳細は，「8.2　メッセージ一覧」を参照してください。

- 環境変数 DCMQCCLTPORT を設定して，リソースアダプタのトランザクションサポートのレベルに XATransaction を指定する場合は，MQC リスナサーバ通信環境定義の mqcenv 定義コマンドの要求受信監視タイマ値（-r オプション）で「監視しない (0)」を設定してください。設定しない場合，MQCONN 命令で MQRC_UNEXPECTED_ERROR（2195）のリターンコードが返され，MQC リスナサーバとの接続に失敗することがあります。
  ただし，要求受信監視タイマ値（-r オプション）で「監視しない (0)」を設定した場合は，検知できない回線障害が発生したときに，MQC ゲートウェイサーバが使用できなくなるため，この件も考慮したアプリケーションを設計してください。

BytesMessage インタフェース（JMS）

BytesMessage インタフェースは，解釈されていないバイトストリームを含むメッセージを送信するために使用します。

形式

```
public interface BytesMessage
extends Message
{
  public boolean readBoolean() throws JMSException;
  public byte readByte() throws JMSException;
  public int readUnsignedByte() throws JMSException;
  public short readShort() throws JMSException;
  public int readUnsignedShort() throws JMSException;
  public char readChar() throws JMSException;
  public int readInt() throws JMSException;
  public long readLong() throws JMSException;
  public float readFloat() throws JMSException;
  public double readDouble() throws JMSException;
  public java.lang.String readUTF() throws JMSException;
  public int readBytes(byte[] value) throws JMSException;
  public int readBytes(byte[] value, int length) throws
JMSException;
  public void writeBoolean(boolean value) throws JMSException;
  public void writeByte(byte value) throws JMSException;
  public void writeShort(short value) throws JMSException;
  public void writeChar(char value) throws JMSException;
  public void writeInt(int value) throws JMSException;
  public void writeLong(long value) throws JMSException;
  public void writeFloat(float value) throws JMSException;
  public void writeDouble(double value) throws JMSException;
  public void writeUTF(java.lang.String value) throws JMSException;
  public void writeBytes(byte[] value) throws JMSException;
  public void writeBytes(byte[] value, int offset, int length)
throws JMSException;
  public void writeObject(java.lang.Object value) throws
JMSException;
  public void reset() throws JMSException;
}
```

メソッド

● public boolean readBoolean() throws JMSException

バイトメッセージストリームから boolean 型の値を取得します。

例外

JMSException：エラーが発生しました。

MessageNotReadableException：メッセージは書き込み専用です。

MessageEOFException：メッセージの終端に達しました。

戻り値：boolean 型の値

● public byte readByte() throws JMSException

バイトメッセージストリームから符号付き 8 ビット型の値を取得します。

例外

JMSException：エラーが発生しました。
MessageNotReadableException：メッセージは書き込み専用です。
MessageEOFException：メッセージの終端に達しました。

戻り値：次の 1 バイトを符号付き 8 ビット型と解釈した値

● public int readUnsignedByte() throws JMSException

バイトメッセージストリームから符号なし 8 ビット型の値を取得します。

例外

JMSException：エラーが発生しました。
MessageNotReadableException：メッセージは書き込み専用です。
MessageEOFException：メッセージの終端に達しました。

戻り値：次の 1 バイトを符号なし 8 ビット型と解釈した値

● public short readShort() throws JMSException

バイトメッセージストリームから符号付き 16 ビット型の値を取得します。

例外

JMSException：エラーが発生しました。
MessageNotReadableException：メッセージは書き込み専用です。
MessageEOFException：メッセージの終端に達しました。

戻り値：次の 2 バイトを符号付き 16 ビット型と解釈した値

● public int readUnsignedShort() throws JMSException

バイトメッセージストリームから符号なし 16 ビット型の値を取得します。

例外

JMSException：エラーが発生しました。
MessageNotReadableException：メッセージは書き込み専用です。
MessageEOFException：メッセージの終端に達しました。

戻り値：次の 2 バイトを符号なし 16 ビット型と解釈した値

● public char readChar() throws JMSException

バイトメッセージストリームから Unicode 文字列を取得します。

例外

JMSException：エラーが発生しました。
MessageNotReadableException：メッセージは書き込み専用です。
MessageEOFException：メッセージの終端に達しました。

戻り値：次の 2 バイトを Unicode 文字列と解釈した値

- public int readInt() throws JMSException

バイトメッセージストリームから符号付き 32 ビット型の値を取得します。

例外

JMSException：エラーが発生しました。
MessageNotReadableException：メッセージは書き込み専用です。
MessageEOFException：メッセージの終端に達しました。

戻り値：次の 4 バイトを符号付き 32 ビット型と解釈した値

- public long readLong() throws JMSException

バイトメッセージストリームから符号付き 64 ビット型の値を取得します。

例外

JMSException：エラーが発生しました。
MessageNotReadableException：メッセージは書き込み専用です。
MessageEOFException：メッセージの終端に達しました。

戻り値：次の 8 バイトを符号付き 64 ビット型と解釈した値

- public float readFloat() throws JMSException

バイトメッセージストリームから float 型の値を取得します。

例外

JMSException：エラーが発生しました。
MessageNotReadableException：メッセージは書き込み専用です。
MessageEOFException：メッセージの終端に達しました。

戻り値：次の 4 バイトを float 型と解釈した値

- public double readDouble() throws JMSException

バイトメッセージストリームから double 型の値を取得します。

例外

JMSException：エラーが発生しました。
MessageNotReadableException：メッセージは書き込み専用です。
MessageEOFException：メッセージの終端に達しました。

戻り値：次の 8 バイトを double 型と解釈した値

- public java.lang.String readUTF() throws JMSException

バイトメッセージストリームから修正 UTF-8 フォーマットを使用してエンコードされた文字列を取得します。

例外

JMSException：エラーが発生しました。
MessageNotReadableException：メッセージは書き込み専用です。
MessageEOFException：メッセージの終端に達しました。

戻り値：Unicode 文字列と解釈した値

- public int readBytes(byte[] value) throws JMSException

バイトメッセージストリームから byte 型配列の値を取得します。

バイトメッセージストリームから読み込まれる残りのバイトより，配列の長さが小さい場合，配列は満たされます。この場合は，あとの呼び出しで次の増分が読み込まれます。一方，バイトメッセージストリームから読み込まれる残りのバイトより，配列の長さが大きい場合，残りのバイトデータは配列に読み込まれます。

戻り値が byte 型配列の長さより小さい場合は，バイトメッセージストリームにデータが残っていないことを示します。この場合は，次の読み込みで -1 が返ります。

パラメタ

value
　バイトメッセージストリームから読み込んだデータを指定します。

例外

JMSException：エラーが発生しました。
MessageNotReadableException：メッセージは書き込み専用です。

戻り値：バッファから入力した総バイト数
ストリームの終端に達して，これ以上データがない場合は -1 が返ります。

- public int readBytes(byte[] value, int length) throws JMSException

バイトメッセージストリームから byte 型配列の値の一部を取得します。

バイトメッセージストリームから読み込まれる残りのバイトより length が小さい場合，読み込まれたデータは配列に length の長さまで格納されます。この場合は，あとの呼び出しで次の増分が読み込まれます。一方，バイトメッセージストリームから読み込まれる残りのバイトより length が大きい場合，残りのバイトデータは配列に読み込まれます。

戻り値が length より小さい場合は，バイトメッセージストリームにデータが残っていな

いことを示します。この場合は，次の読み込みで -1 が返ります。

length の値が負の場合，または配列の長さより大きい場合は，例外 JMSException がスローされます。例外がスローされると，バイトメッセージストリームからデータは読み込まれません。

パラメタ

value

バイトメッセージストリームから読み込んだデータを指定します。

length

読み込むデータのバイト長を指定します。配列の長さと等しい，または小さい値を指定してください。

例外

JMSException：エラーが発生しました。
MessageNotReadableException：メッセージは書き込み専用です。

戻り値：バッファから入力した総バイト数
ストリームの終端に達して，これ以上データがない場合は -1 が返ります。

● public void writeBoolean(boolean value) throws JMSException

1 バイトの値として，boolean 型の値をバイトメッセージストリームに書き込みます。true の場合は 1 の値，false の場合は 0 の値を書き込みます。

パラメタ

value

書き込む値を指定します。

例外

JMSException：エラーが発生しました。
MessageNotWriteableException：メッセージは読み込み専用です。

● public void writeByte(byte value) throws JMSException

1 バイトの値として，byte 型の値をバイトメッセージストリームに書き込みます。

パラメタ

value

書き込む値を指定します。

例外

JMSException：エラーが発生しました。
MessageNotWriteableException：メッセージは読み込み専用です。

● public void writeShort(short value) throws JMSException

2 バイトの値として，short 型の値を high byte first でバイトメッセージストリームに書き込みます。

パラメタ

value
書き込む値を指定します。

例外

JMSException：エラーが発生しました。
MessageNotWriteableException：メッセージは読み込み専用です。

● public void writeChar(char value) throws JMSException

2 バイトの値として，char 型の値を high byte first でバイトメッセージストリームに書き込みます。

パラメタ

value
書き込む値を指定します。

例外

JMSException：エラーが発生しました。
MessageNotWriteableException：メッセージは読み込み専用です。

● public void writeInt(int value) throws JMSException

4 バイトの値として，int 型の値を high byte first でバイトメッセージストリームに書き込みます。

パラメタ

value
書き込む値を指定します。

例外

JMSException：エラーが発生しました。
MessageNotWriteableException：メッセージは読み込み専用です。

● public void writeLong(long value) throws JMSException

8 バイトの値として，long 型の値を high byte first でバイトメッセージストリームに書き込みます。

パラメタ

value
書き込む値を指定します。

例外

JMSException：エラーが発生しました。
MessageNotWriteableException：メッセージは読み込み専用です。

● public void writeFloat(float value) throws JMSException

Float クラスの floatToIntBits メソッドを使用して，float 型の値を int 型に変換します。その後，4 バイトの値として，変換した int 型の値を high byte first でバイトメッセージストリームに書き込みます。

パラメタ

value
書き込む値を指定します。

例外

JMSException：エラーが発生しました。
MessageNotWriteableException：メッセージは読み込み専用です。

● public void writeDouble(double value) throws JMSException

Double クラスの doubleToLongBits メソッドを使用して，double 型の値を long 型に変換します。その後，8 バイトの値として，変換した long 型の値を high byte first でバイトメッセージストリームに書き込みます。

パラメタ

value
書き込む値を指定します。

例外

JMSException：エラーが発生しました。
MessageNotWriteableException：メッセージは読み込み専用です。

● public void writeUTF(java.lang.String value) throws JMSException

修正 UTF-8 フォーマットを使用してエンコードされた文字列をバイトメッセージストリームに書き込みます。

パラメタ

value
書き込む値を指定します。

例外

JMSException：エラーが発生しました。
MessageNotWriteableException：メッセージは読み込み専用です。

● public void writeBytes(byte[] value) throws JMSException

byte 型配列をバイトメッセージストリームに書き込みます。

パラメタ

value

書き込む値を指定します。

例外

JMSException：エラーが発生しました。

MessageNotWriteableException：メッセージは読み込み専用です。

● public void writeBytes(byte[] value, int offset, int length) throws JMSException

byte 型配列の一部をバイトメッセージストリームに書き込みます。

パラメタ

value

書き込む値を指定します。

offset

byte 型配列内の初期のオフセットを指定します。

length

書き込むバイト数を指定します。

例外

JMSException：エラーが発生しました。

MessageNotWriteableException：メッセージは読み込み専用です。

● public void writeObject(java.lang.Object value) throws JMSException

Java オブジェクトをバイトメッセージストリームに書き込みます。書き込みできる値の型は，プリミティブ型をラップするクラス（Byte，Integer など），String 型，および byte 型配列です。

パラメタ

value

書き込む Java オブジェクトの値を指定します。null 値は指定できません。

例外

JMSException：エラーが発生しました。

MessageNotWriteableException：メッセージは読み込み専用です。

NullPointerException：パラメタが null 値です。

MessageFormatException：オブジェクトの型が不正です。

● public void reset() throws JMSException

メッセージ本体を読み込み専用にして，バイトメッセージストリームを最初の位置に変更します。

例外

JMSException：エラーが発生しました。

MessageFormatException：メッセージのフォーマットが不正です。

ConnectionMetaData インタフェース（JMS）

ConnectionMetaData オブジェクトは，JMS の基本情報を提供します。このオブジェクトは，QueueConnection 内の getMetaData メソッドによって生成されます。

ConnectionMetaData の機能を次に示します。

- JMS サポートバージョンの返却
- JMS プロバイダ名称の返却
- JMS プロバイダバージョンの返却

形式

```
public interface ConnectionMetaData
{
  public int getJMSMajorVersion() throws JMSException;
  public int getJMSMinorVersion() throws JMSException;
  public java.lang.String getJMSProviderName() throws JMSException;
  public java.lang.String getJMSVersion() throws JMSException;
  public java.util.Enumeration getJMSXPropertyNames() throws
JMSException;
  public int getProviderMajorVersion() throws JMSException;
  public int getProviderMinorVersion() throws JMSException;
  public java.lang.String getProviderVersion() throws JMSException;
}
```

メソッド

● public int getJMSMajorVersion() throws JMSException

対応する JMS インタフェースのメジャーバージョンを返します。

例外

JMSException：該当しません。

戻り値：対応する JMS インタフェースのメジャーバージョン

● public int getJMSMinorVersion() throws JMSException

対応する JMS インタフェースのマイナーバージョンを返します。

例外

JMSException：該当しません。

戻り値：対応する JMS インタフェースのマイナーバージョン

● public java.lang.String getJMSProviderName() throws JMSException

プロバイダ名称を返します。

例外

JMSException：該当しません。

戻り値：プロバイダ名称

- public java.lang.String getJMSVersion() throws JMSException

対応する JMS インタフェースのバージョンを返します。

例外

JMSException：該当しません。

戻り値：対応する JMS インタフェースのバージョン

- public java.util.Enumeration getJMSXPropertyNames() throws JMSException

JMSX プロパティ名の列挙を返します。

例外

JMSException：該当しません。

戻り値：JMSX プロパティ名の列挙

- public int getProviderMajorVersion() throws JMSException

TP1/Message Queue - Access の JMS インタフェースに対するメジャーバージョンを返します。

例外

JMSException：該当しません。

戻り値：TP1/Message Queue - Access の JMS インタフェースに対するメジャーバージョン

- public int getProviderMinorVersion() throws JMSException

TP1/Message Queue - Access の JMS インタフェースに対するマイナーバージョンを返します。

例外

JMSException：該当しません。

戻り値：TP1/Message Queue - Access の JMS インタフェースに対するマイナーバージョン

- public java.lang.String getProviderVersion() throws JMSException

TP1/Message Queue - Access の JMS インタフェースに対するバージョンを返します。

例外

JMSException：該当しません。

戻り値：TP1/Message Queue - Access の JMS インタフェースに対するバージョン

DeliveryMode インタフェース（JMS）

DeliveryMode インタフェースは，JMS でサポートしているメッセージの配送モードに基づき，メッセージの配送モードを表す定数を定義するインタフェースです。この定数は，TP1/Message Queue の MQMD 構造体の Persistence（メッセージの永続性）フィールドに格納される値に対応します。

形式

```
public interface DeliveryMode
{
  public static final int NON_PERSISTENT;
  public static final int PERSISTENT;
}
```

フィールド

● public static final int NON_PERSISTENT

非永続メッセージを表す定数です。TP1/Message Queue では MQPER_NOT_PERSISTENT に相当します。

非永続メッセージ（NON_PERSISTENT）の場合，補助記憶装置に記録する必要がないためオーバヘッドが小さくなります。その代わり，送信を保証されるのは 1 回だけです。メッセージが失われた場合も，2 回送信されることはありません。

● public static final int PERSISTENT

永続メッセージを表す定数です。TP1/Message Queue では MQPER_PERSISTENT に相当します。

クライアントが send メソッドを発行すると，TP1/Message Queue では補助記憶装置にメッセージを記録します。したがって，永続メッセージ（PERSISTENT）が失われるのは，ハードウェア障害の場合だけです。

Destination インタフェース・Queue インタフェース（JMS）

Destination インタフェースは，送受信するメッセージの宛先のソースを指定するための，JMS 管理オブジェクトです。

Queue インタフェースは，メッセージの宛先となるキューをカプセル化したものです。

形式

```
public interface Destination
{ }

public interface Queue
extends Destination
{
  public java.lang.String getQueueName() throws JMSException;
  public java.lang.String toString();
}
```

メソッド

- public java.lang.String getQueueName() throws JMSException

キューの名称を取得します。名前に依存したクライアントにはポータビリティが保証されません。

例外

JMSException：エラーが発生しました。

戻り値：キュー名称

- public java.lang.String toString()

出力されたキュー名称を返します。java.lang.Object の toString メソッドをオーバライドします。

戻り値：プロバイダがこのキューを特定する識別用の値

Message インタフェース（JMS）

Message インタフェースは，すべての JMS メッセージのルートインタフェースです。

すべてのメッセージのために使用される JMS ヘッダを定義します。ヘッダと本体から構成される小容量のエンティティとしてメッセージを扱います。メッセージ本体は，送られるアプリケーションデータを含んでいます。

形式

```
public interface Message
{
  public static final int DEFAULT_DELIVERY_MODE;
  public static final int DEFAULT_PRIORITY;
  public static final long DEFAULT_TIME_TO_LIVE;
  public java.lang.String getJMSMessageID() throws JMSException;
  public void setJMSMessageID(java.lang.String id) throws
JMSException;
  public long getJMSTimestamp() throws JMSException;
  public void setJMSTimestamp(long timestamp) throws JMSException;
  public byte[] getJMSCorrelationIDAsBytes() throws JMSException;
  public void setJMSCorrelationIDAsBytes(byte[] correlationID)
throws JMSException;
  public java.lang.String getJMSCorrelationID() throws
JMSException;
  public void setJMSCorrelationID(java.lang.String correlationID)
throws JMSException;
  public Destination getJMSReplyTo() throws JMSException;
  public void setJMSReplyTo(Destination replyTo) throws
JMSException;
  public Destination getJMSDestination() throws JMSException;
  public void setJMSDestination(Destination destination) throws
JMSException;
  public int getJMSDeliveryMode() throws JMSException;
  public void setJMSDeliveryMode(int deliveryMode) throws
JMSException;
  public boolean getJMSRedelivered() throws JMSException;
  public void setJMSRedelivered(boolean redelivered) throws
JMSException;
  public java.lang.String getJMSType() throws JMSException;
  public void setJMSType(java.lang.String type) throws JMSException;
  public long getJMSExpiration() throws JMSException;
  public void setJMSExpiration(long expiration) throws JMSException;
  public int getJMSPriority() throws JMSException;
  public void setJMSPriority(int priority) throws JMSException;
  public void clearProperties() throws JMSException;
  public boolean propertyExists(java.lang.String name) throws
JMSException;
  public boolean getBooleanProperty(java.lang.String name) throws
JMSException;
  public byte getByteProperty(java.lang.String name) throws
JMSException;
  public short getShortProperty(java.lang.String name) throws
JMSException;
  public int getIntProperty(java.lang.String name) throws
```

```
JMSException;
  public long getLongProperty(java.lang.String name) throws
JMSException;
  public float getFloatProperty(java.lang.String name) throws
JMSException;
  public double getDoubleProperty(java.lang.String name) throws
JMSException;
  public java.lang.String getStringProperty(java.lang.String name)
throws JMSException;
  public java.lang.Object getObjectProperty(java.lang.String name)
throws JMSException;
  public java.util.Enumeration getPropertyNames() throws
JMSException;
  public void setBooleanProperty(java.lang.String name, boolean
value) throws JMSException;
  public void setByteProperty(java.lang.String name, byte value)
throws JMSException;
  public void setShortProperty(java.lang.String name, short value)
throws JMSException;
  public void setIntProperty(java.lang.String name, int value)
throws JMSException;
  public void setLongProperty(java.lang.String name, long value)
throws JMSException;
  public void setFloatProperty(java.lang.String name, float value)
throws JMSException;
  public void setDoubleProperty(java.lang.String name, double value)
throws JMSException;
  public void setStringProperty(java.lang.String name,
java.lang.String value) throws JMSException;
  public void setObjectProperty(java.lang.String name,
java.lang.Object value) throws JMSException;
  public void acknowledge() throws JMSException;
  public void clearBody() throws JMSException;
}
```

フィールド

- public static final int DEFAULT_DELIVERY_MODE

メッセージの永続性のデフォルトです。デフォルト値は DeliveryMode.PERSISTENT（永続）です。

- public static final int DEFAULT_PRIORITY

メッセージの優先度のデフォルトです。デフォルト値は 4 です。

- public static final long DEFAULT_TIME_TO_LIVE

メッセージの保持時間のデフォルトです。デフォルト値は 0 です。

メソッド

- public java.lang.String getJMSMessageID() throws JMSException

JMSMessageID ヘッダフィールドを取得します。

例外

JMSException：エラーが発生しました。

戻り値：メッセージ識別子

● public void setJMSMessageID(java.lang.String id) throws JMSException

JMSMessageID ヘッダフィールドを設定します。JMSMessageID ヘッダフィールドは，送信時に TP1/Message Queue によって設定されるため，ユーザが設定しても無視されます。

受信したメッセージの JMSMessageID ヘッダフィールドを変更するときにこのメソッドを使用します。

JMSMessageID の先頭には必ず「ID:」を付けてください。

パラメタ

id

メッセージ識別子を指定します。

例外

JMSException：エラーが発生しました。

● public long getJMSTimestamp() throws JMSException

JMSTimestamp ヘッダフィールドを取得します。

例外

JMSException：エラーが発生しました。

戻り値：ミリ秒で表される時刻

協定世界時の 1970 年 1 月 1 日午前 0 時から経過したミリ秒の値です。

● public void setJMSTimestamp(long timestamp) throws JMSException

JMSTimestamp ヘッダフィールドを設定します。TP1/Message Queue は，送信時に PutDate，PutTime に対して時刻を設定するため，JMSTimestamp ヘッダフィールドにユーザが値を設定しても無視されます。

JMSTimestamp は，TP1/Message Queue に対してメッセージのキューへの格納を依頼した時点の時刻です。メッセージが実際に転送先に送られる時刻ではありません。

受信したメッセージの JMSTimestamp を変更するときにこのメソッドを使用します。

パラメタ

timestamp

ミリ秒で表される時刻を指定します。

協定世界時の 1970 年 1 月 1 日午前 0 時から経過したミリ秒の値を指定してください。

例外

JMSException：エラーが発生しました。

● public byte[] getJMSCorrelationIDAsBytes() throws JMSException

相関識別子を byte 型配列で取得します。

例外

JMSException：エラーが発生しました。

戻り値：相関識別子

● public void setJMSCorrelationIDAsBytes(byte[] correlationID) throws JMSException

相関識別子を byte 型配列で設定します。24 バイトを超える部分は切り捨てられます。

パラメタ

correlationID

相関識別子を指定します。

例外

JMSException：エラーが発生しました。

● public java.lang.String getJMSCorrelationID() throws JMSException

相関識別子を String 型で取得します。

例外

JMSException：エラーが発生しました。

戻り値：相関識別子

● public void setJMSCorrelationID(java.lang.String correlationID) throws JMSException

相関識別子を String 型で設定します。

相関識別子には次の値のどちらかを設定します。

- プロバイダ固有のメッセージ識別子
  先頭が「ID:」で，以降が 48 けたの 16 進数の文字列で指定します。内部で，長さ 24 バイトの byte 型配列（TP1/Message Queue の MQBYTE24 型）に変換されます。
- アプリケーション固有の文字列
  先頭が「ID:」ではない文字列で指定します。内部で String 型から MQBYTE24 型に変換されます。このとき，24 バイトを超える部分は切り捨てられます。
  また，メッセージの送信を契機に，JMSCorrelationID に格納されたアプリケーション固有の文字列は，先頭が「ID:」で以降が 48 けたの 16 進数の文字列に置き換えられます。

パラメタ

correlationID
相関識別子を指定します。

例外
JMSException：エラーが発生しました。

● public Destination getJMSReplyTo() throws JMSException

応答キューを取得します。

例外
JMSException：エラーが発生しました。

戻り値：応答キュー

● public void setJMSReplyTo(Destination replyTo) throws JMSException

応答キューを設定します。

パラメタ

replyTo
応答キューを指定します。

例外
JMSException：エラーが発生しました。

● public Destination getJMSDestination() throws JMSException

宛先キューを取得します。

例外
JMSException：エラーが発生しました。

戻り値：宛先キュー

● public void setJMSDestination(Destination destination) throws JMSException

宛先キューを設定します。ただし，JMSDestination ヘッダフィールドに値を設定しても，送信時には無視されます。送信完了後，JMSDestination ヘッダフィールドには送信メソッドによって指定される宛先オブジェクトが格納されます。受信時は，JMSDestination ヘッダフィールドには，該当するメッセージが入っていたキューが設定されます。

パラメタ

destination
宛先キューを指定します。

例外

JMSException：エラーが発生しました。

● public int getJMSDeliveryMode() throws JMSException

メッセージの永続性を取得します。

例外

JMSException：エラーが発生しました。

戻り値

DeliveryMode.NON_PERSISTENT：非永続
DeliveryMode.PERSISTENT：永続

● public void setJMSDeliveryMode(int deliveryMode) throws JMSException

メッセージの永続性を設定します。ただし，JMSDeliveryMode ヘッダフィールドに値を設定しても，送信時には無視されます。メッセージの永続性は QueueSender インタフェースで設定します。

パラメタ

deliveryMode

メッセージの永続性を次の値で指定します。
DeliveryMode.NON_PERSISTENT：非永続
DeliveryMode.PERSISTENT：永続

例外

JMSException：エラーが発生しました。

● public boolean getJMSRedelivered() throws JMSException

このメッセージが再送されているかどうかを取得します。

例外

JMSException：エラーが発生しました。

戻り値

false：MQMD 構造体の BackoutCount フィールドの値が 0
true：MQMD 構造体の BackoutCount フィールドの値が 1 以上

● public void setJMSRedelivered(boolean redelivered) throws JMSException

再送のフラグを設定します。ただし，JMSRedelivered ヘッダフィールドに値を設定しても，送信時には無視されます。

パラメタ

redelivered

再送のフラグを指定します。

例外

JMSException：エラーが発生しました。

● public java.lang.String getJMSType() throws JMSException

TP1/Message Queue は JMSType の値を送受信できないため，setJMSType メソッドで設定された値がそのまま返ります。

例外

JMSException：エラーが発生しました。

戻り値：JMSType 文字列

● public void setJMSType(java.lang.String type) throws JMSException

JMSType ヘッダフィールドを設定しますが，TP1/Message Queue のキューにはこの情報は記録されません。

例外

JMSException：エラーが発生しました。

● public long getJMSExpiration() throws JMSException

メッセージの満了時刻を取得します。

メッセージの満了時刻（JMSExpiration）は，メッセージの登録時刻にメッセージの保持時間（TimeToLive）を加算した時刻に相当します。

なお，メッセージの満了時刻（JMSExpiration）が 0 の場合，メッセージが満了しないことを意味します。

例外

JMSException：エラーが発生しました。

戻り値：メッセージの満了時刻

● public void setJMSExpiration(long expiration) throws JMSException

メッセージの満了時刻を設定します。ただし，JMSExpiration ヘッダフィールドに値を設定しても，送信時には無視されます。TimeToLive（メッセージの保持時間）は QueueSender インタフェースで設定します。TimeToLive に 0 を指定した場合，JMSExpiration ヘッダフィールドは 0 に設定され，メッセージが満了しないことを示します。

パラメタ

expiration

メッセージの満了時刻（単位：ミリ秒）を指定します。0 の場合，メッセージは満了しません。

例外

JMSException：エラーが発生しました。

● public int getJMSPriority() throws JMSException

メッセージの優先度を取得します。

例外

JMSException：エラーが発生しました。

戻り値：メッセージの優先度

● public void setJMSPriority(int priority) throws JMSException

メッセージの優先度を設定します。ただし，JMSPriority ヘッダフィールドに値を設定しても，送信時には無視されます。メッセージの優先度は QueueSender インタフェースで設定します。

パラメタ

priority

メッセージの優先度を指定します。

例外

JMSException：エラーが発生しました。

● public void clearProperties() throws JMSException

プロパティの値を初期化します。ヘッダや本体は削除されません。

例外

JMSException：エラーが発生しました。

● public boolean propertyExists(java.lang.String name) throws JMSException

プロパティ（名称：name）の値が存在するかどうかを調べます。

パラメタ

name

プロパティの名称を指定します。

例外

JMSException：エラーが発生しました。

戻り値

true：プロパティの値が存在します。

false：プロパティの値が存在しません。

● public boolean getBooleanProperty(java.lang.String name) throws JMSException

boolean 型のプロパティ（名称：name）の値を取得します。

パラメタ

name

プロパティの名称を指定します。

例外

JMSException：エラーが発生しました。

MessageFormatException：プロパティ値の型変換が不正です。

戻り値：プロパティ name の値

● public byte getByteProperty(java.lang.String name) throws JMSException

byte 型のプロパティ（名称：name）の値を取得します。

パラメタ

name

プロパティの名称を指定します。

例外

JMSException：エラーが発生しました。

MessageFormatException：プロパティ値の型変換が不正です。

戻り値：プロパティ name の値

● public short getShortProperty(java.lang.String name) throws JMSException

short 型のプロパティ（名称：name）の値を取得します。

パラメタ

name

プロパティの名称を指定します。

例外

JMSException：エラーが発生しました。

MessageFormatException：プロパティ値の型変換が不正です。

戻り値：プロパティ name の値

● public int getIntProperty(java.lang.String name) throws JMSException

int 型のプロパティ（名称：name）の値を取得します。

パラメタ

name

プロパティの名称を指定します。

例外

JMSException：エラーが発生しました。

MessageFormatException：プロパティ値の型変換が不正です。

戻り値：プロパティ name の値

● public long getLongProperty(java.lang.String name) throws JMSException

long 型のプロパティ（名称：name）の値を取得します。

パラメタ

name

プロパティの名称を指定します。

例外

JMSException：エラーが発生しました。
MessageFormatException：プロパティ値の型変換が不正です。

戻り値：プロパティ name の値

● public float getFloatProperty(java.lang.String name) throws JMSException

float 型のプロパティ（名称：name）の値を取得します。

パラメタ

name

プロパティの名称を指定します。

例外

JMSException：エラーが発生しました。
MessageFormatException：プロパティ値の型変換が不正です。

戻り値：プロパティ name の値

● public double getDoubleProperty(java.lang.String name) throws JMSException

double 型のプロパティ（名称：name）の値を取得します。

パラメタ

name

プロパティの名称を指定します。

例外

JMSException：エラーが発生しました。
MessageFormatException：プロパティ値の型変換が不正です。

戻り値：プロパティ name の値

● public java.lang.String getStringProperty(java.lang.String name) throws JMSException

String 型のプロパティ（名称：name）の値を取得します。

パラメタ

name

プロパティの名称を指定します。

例外

JMSException：エラーが発生しました。

MessageFormatException：プロパティ値の型変換が不正です。

戻り値：プロパティ name の値

● public java.lang.Object getObjectProperty(java.lang.String name) throws JMSException

プロパティ（名称：name）の値をプリミティブ型をラップするクラスのオブジェクトとして取得します。

パラメタ

name

プロパティの名称を指定します。

例外

JMSException：エラーが発生しました。

戻り値：プロパティ name の値

● public java.util.Enumeration getPropertyNames() throws JMSException

すべてのプロパティの名称を取得します。ヘッダは返されません。

例外

JMSException：エラーが発生しました。

戻り値：プロパティの一覧

● public void setBooleanProperty(java.lang.String name, boolean value) throws JMSException

boolean 型のプロパティ（名称：name）の値を設定します。

name に指定するプロパティ名が，「JMS メッセージのヘッダとプロパティ」の表 7-6 に示すプロパティ名ではない場合で send メソッドの発行時には，プロパティの値は TP1/Message Queue のキューには格納されません。

パラメタ

name

プロパティの名称を指定します。

value
プロパティの値を指定します。

例外
JMSException：エラーが発生しました。
MessageNotWriteableException：プロパティは読み込み専用です。

● public void setByteProperty(java.lang.String name, byte value) throws JMSException

byte 型のプロパティ（名称：name）の値を設定します。

name に指定するプロパティ名が，「JMS メッセージのヘッダとプロパティ」の表 7-6 に示すプロパティ名ではない場合で send メソッドの発行時には，プロパティの値は TP1/Message Queue のキューには格納されません。

パラメタ
name
プロパティの名称を指定します。
value
プロパティの値を指定します。

例外
JMSException：エラーが発生しました。
MessageNotWriteableException：プロパティは読み込み専用です。

● public void setShortProperty(java.lang.String name, short value) throws JMSException

short 型のプロパティ（名称：name）の値を設定します。

name に指定するプロパティ名が，「JMS メッセージのヘッダとプロパティ」の表 7-6 に示すプロパティ名ではない場合で send メソッドの発行時には，プロパティの値は TP1/Message Queue のキューには格納されません。

パラメタ
name
プロパティの名称を指定します。
value
プロパティの値を指定します。

例外
JMSException：エラーが発生しました。
MessageNotWriteableException：プロパティは読み込み専用です。

● public void setIntProperty(java.lang.String name, int value) throws JMSException

int 型のプロパティ（名称：name）の値を設定します。

name に指定するプロパティ名が，「JMS メッセージのヘッダとプロパティ」の表 7-6 に示すプロパティ名ではない場合で send メソッドの発行時には，プロパティの値は TP1/Message Queue のキューには格納されません。

パラメタ

name
　プロパティの名称を指定します。

value
　プロパティの値を指定します。

例外

JMSException：エラーが発生しました。
MessageNotWriteableException：プロパティは読み込み専用です。

● public void setLongProperty(java.lang.String name, long value) throws JMSException

long 型のプロパティ（名称：name）の値を設定します。

name に指定するプロパティ名が，「JMS メッセージのヘッダとプロパティ」の表 7-6 に示すプロパティ名ではない場合で send メソッドの発行時には，プロパティの値は TP1/Message Queue のキューには格納されません。

パラメタ

name
　プロパティの名称を指定します。

value
　プロパティの値を指定します。

例外

JMSException：エラーが発生しました。
MessageNotWriteableException：プロパティは読み込み専用です。

● public void setFloatProperty(java.lang.String name, float value) throws JMSException

float 型のプロパティ（名称：name）の値を設定します。

name に指定するプロパティ名が，「JMS メッセージのヘッダとプロパティ」の表 7-6 に示すプロパティ名ではない場合で send メソッドの発行時には，プロパティの値は TP1/Message Queue のキューには格納されません。

パラメタ

name
　プロパティの名称を指定します。

value
　プロパティの値を指定します。

例外

JMSException：エラーが発生しました。
MessageNotWriteableException：プロパティは読み込み専用です。

● public void setDoubleProperty(java.lang.String name, double value) throws JMSException

double 型のプロパティ（名称：name）の値を設定します。

name に指定するプロパティ名が，「JMS メッセージのヘッダとプロパティ」の表 7-6 に示すプロパティ名ではない場合で send メソッドの発行時には，プロパティの値は TP1/Message Queue のキューには格納されません。

パラメタ

name
　プロパティの名称を指定します。

value
　プロパティの値を指定します。

例外

JMSException：エラーが発生しました。
MessageNotWriteableException：プロパティは読み込み専用です。

● public void setStringProperty(java.lang.String name, java.lang.String value) throws JMSException

String 型のプロパティ（名称：name）の値を設定します。

name に指定するプロパティ名が，「JMS メッセージのヘッダとプロパティ」の表 7-6 に示すプロパティ名ではない場合で send メソッドの発行時には，プロパティの値は TP1/Message Queue のキューには格納されません。

パラメタ

name
　プロパティの名称を指定します。

value
　プロパティの値を指定します。

例外

JMSException：エラーが発生しました。
MessageNotWriteableException：プロパティは読み込み専用です。

● public void setObjectProperty(java.lang.String name, java.lang.Object value) throws JMSException

プロパティ（名称：name）の値を設定します。設定できる値の型は，プリミティブ型を

ラップするクラス（Byte，Integer など），および String 型です。

name に指定するプロパティ名が，「JMS メッセージのヘッダとプロパティ」の表 7-6 に示すプロパティ名ではない場合で send メソッドの発行時には，プロパティの値は TP1/Message Queue のキューには格納されません。

パラメタ

name
　プロパティの名称を指定します。

value
　プロパティの値を指定します。

例外

JMSException：エラーが発生しました。
MessageFormatException：プロパティ値の型が不正です。
MessageNotWriteableException：プロパティは読み込み専用です。

● public void acknowledge() throws JMSException

現時点から以前のすべてのメッセージを確認します。ただし，確認機能は提供しません。このメソッドを発行しても，即座にリターンします。

すべての JMS メッセージは，クライアントが「受信側でメッセージは明示的に確認されなければならない」と指定しているときに使われる，acknowledge メソッドをサポートしています。クライアントが自動確認機能を使用している場合，acknowledge メソッドの呼び出しは無視されます。

このメソッドを発行した場合，受け取られたにもかかわらず確認されていないメッセージは，受信側に再送される場合があります。

例外

JMSException：エラーが発生しました。
IllegalStateException：すでにクローズされたセションでこのメソッドが発行されました。

● public void clearBody() throws JMSException

メッセージ本体を削除します。ヘッダやプロパティは削除されません。

メッセージ本体が読み込み専用の場合，メッセージ本体は削除されるとともに書き込み専用となります。

例外

JMSException：エラーが発生しました。

MessageConsumer インタフェース・QueueReceiver インタフェース（JMS）

QueueReceiver インタフェースは，クライアントがキューからメッセージを受け取るために使用します。

形式

```
public interface MessageConsumer
{
  public void close() throws JMSException;
  public MessageListener getMessageListener() throws JMSException;
  public java.lang.String getMessageSelector() throws JMSException;
  public Message receive() throws JMSException;
  public Message receive(long timeout) throws JMSException;
  public Message receiveNoWait() throws JMSException;
  public void setMessageListener(MessageListener listener) throws
JMSException;
}

public interface QueueReceiver
extends MessageConsumer
{
  public Queue getQueue() throws JMSException;
}
```

メソッド

- public void close() throws JMSException

TP1/Message Queue が確保した資源を解放するために，資源（キュー）をクローズします。

例外

JMSException：クローズに失敗しました。

- public MessageListener getMessageListener() throws JMSException

メッセージリスナを返します。

例外

JMSException：エラーが発生しました。

戻り値：メッセージリスナ

- public java.lang.String getMessageSelector() throws JMSException

メッセージセレクタを返します。

例外

JMSException：エラーが発生しました。

戻り値：メッセージセレクタ

● public Message receive() throws JMSException

該当するキューレシーバのために作られた，次メッセージを受け取ります。

メッセージが作られるまで，またはこのキューレシーバがクローズされるまで，このメソッドは無期限にブロックされて待ち状態になります。

トランザクション内でこのメソッドが実行された場合，コミットされるまでメッセージは残ります。

コネクションが受信禁止状態の場合は，null が返され，メッセージ KFCA31307-W が出力されます。

例外

JMSException：エラーが発生しました。

戻り値：該当するキューレシーバのために作られた次メッセージ
このキューレシーバが同時にクローズされた場合，またはコネクションが受信禁止状態の場合は，null が返ります。

● public Message receive(long timeout) throws JMSException

指定したタイムアウト時間以内に到着した次メッセージを受け取ります。

メッセージが作られるまで，タイムアウト時間が終了するまで，またはこのキューレシーバがクローズされるまで，このメソッドはブロックされて待ち状態になります。

timeout が 0 の場合は，無期限にブロックされます。

コネクションが受信禁止状態の場合は，null が返され，メッセージ KFCA31307-W が出力されます。

パラメタ

timeout

タイムアウト時間（単位：ミリ秒）を指定します。
0 以上の値を指定します。ただし，1 ～ 999 の値を指定すると，1 秒間ブロックされて待ち状態になります。
0 未満の値を指定した場合，例外 JMSException がスローされます。

例外

JMSException：エラーが発生しました。

戻り値：該当するキューレシーバのために作られた次メッセージ
タイムアウト時間が終了した場合，このキューレシーバが同時にクローズされた場合，またはコネクションが受信禁止状態の場合は，null が返ります。

● public Message receiveNoWait() throws JMSException

次メッセージをすぐに利用できる場合，そのメッセージを受け取ります。

コネクションが受信禁止状態の場合は，null が返され，メッセージ KFCA31307-W が出力されます。

例外

JMSException：エラーが発生しました。

戻り値：該当するキューレシーバのために作られた次メッセージ
次メッセージが利用できない場合，またはコネクションが受信禁止状態の場合は，null が返ります。

● public void setMessageListener(MessageListener listener) throws JMSException

メッセージリスナを設定します。

メッセージリスナは QueueReceiver オブジェクト内で記憶されますが，実際には使用されないので，注意が必要です

パラメタ

listener
メッセージリスナを指定します。

例外

JMSException：エラーが発生しました。

● public Queue getQueue() throws JMSException

キューレシーバに関連づけられたキューのオブジェクトを返します。

例外

JMSException：キューハンドルが取得できませんでした。

戻り値：Queue オブジェクト

MessageProducer インタフェース・QueueSender インタフェース（JMS）

QueueSender インタフェースは，クライアントがキューにメッセージを送るために使用します。

形式

```
public interface MessageProducer
{
  public void close() throws JMSException;
  public int getDeliveryMode() throws JMSException;
  public boolean getDisableMessageID() throws JMSException;
  public boolean getDisableMessageTimestamp() throws JMSException;
  public int getPriority() throws JMSException;
  public long getTimeToLive() throws JMSException;
 public void setDeliveryMode(int deliveryMode) throws JMSException;
  public void setDisableMessageID(boolean value) throws
JMSException;
  public void setDisableMessageTimestamp(boolean value) throws
JMSException;
  public void setPriority(int defaultPriority) throws JMSException;
  public void setTimeToLive(long timeToLive) throws JMSException;
}

public interface QueueSender
extends MessageProducer
{
  public Queue getQueue() throws JMSException;
  public void send(Message message) throws JMSException;
 public void send(Message message, int deliveryMode, int priority,
long timeToLive) throws JMSException;
  public void send(Queue queue, Message message) throws
JMSException;
  public void send(Queue queue, Message message, int deliveryMode,
int priority, long timeToLive) throws JMSException;
}
```

メソッド

● public void close() throws JMSException

TP1/Message Queue が確保した資源を解放するために，資源（キュー）をクローズします。

例外

JMSException：クローズに失敗しました。

● public int getDeliveryMode() throws JMSException

キュー内のメッセージに対する省略時の永続性を取得します。

例外

JMSException：エラーが発生しました。

戻り値

DeliveryMode.NON_PERSISTENT：非永続
DeliveryMode.PERSISTENT：永続

● public boolean getDisableMessageID() throws JMSException

メッセージ識別子が使用不可であるかどうかを取得します。ただし，このメソッドで取得した値に関係なく，TP1/Message Queue では常にメッセージ識別子は使用可能です。

例外

JMSException：エラーが発生しました。

戻り値

true：メッセージ識別子は使用不可
false：メッセージ識別子は使用可能

● public boolean getDisableMessageTimestamp() throws JMSException

登録日時が使用不可であるかどうかを取得します。ただし，このメソッドで取得した値に関係なく，TP1/Message Queue では常に登録日時は使用可能です。

例外

JMSException：エラーが発生しました。

戻り値

true：登録日時は使用不可
false：登録日時は使用可能

● public int getPriority() throws JMSException

キュー内のメッセージに対する省略時の優先度を取得します。

JMS では，0 ～ 9 の 10 段階の優先度を定義しています。TP1/Message Queue では，0 ～ MaxPriority の値をとります。MaxPriority は，キューマネジャで定義されています。

例外

JMSException：エラーが発生しました。

戻り値：キュー内のメッセージに対する省略時の優先度

● public long getTimeToLive() throws JMSException

キュー内のメッセージに対する省略時のメッセージの保持時間を取得します。

例外

JMSException：エラーが発生しました。

戻り値：キュー内のメッセージに対する省略時のメッセージの保持時間（単位：ミリ秒）

● public void setDeliveryMode(int deliveryMode) throws JMSException

キュー内のメッセージに対する省略時の永続性を設定します。デフォルト値は DeliveryMode.PERSISTENT（永続）です。

パラメタ

deliveryMode

メッセージの永続性を次の値で指定します。
DeliveryMode.NON_PERSISTENT：非永続
DeliveryMode.PERSISTENT：永続
上記以外の値を指定した場合，例外 JMSException がスローされます。

例外

JMSException：エラーが発生しました。

● public void setDisableMessageID(boolean value) throws JMSException

メッセージ識別子を使用不可にするかどうかを設定します。デフォルト値は使用可能です。

このメソッドで設定した値に関係なく，TP1/Message Queue では常にメッセージ識別子は使用可能です。

パラメタ

value

メッセージ識別子を使用不可にするかどうかを次の値で指定します。
true：使用不可にします。
false：使用可能にします。

例外

JMSException：エラーが発生しました。

● public void setDisableMessageTimestamp(boolean value) throws JMSException

登録日時を使用不可にするかどうかを設定します。デフォルト値は使用可能です。

このメソッドで設定した値に関係なく，TP1/Message Queue では常に登録日時は使用可能です。

パラメタ

value

登録日時を使用不可にするかどうかを次の値で指定します。
true：使用不可にします。
false：使用可能にします。

例外

JMSException：エラーが発生しました。

● public void setPriority(int defaultPriority) throws JMSException

キュー内のメッセージに対する省略時の優先度を設定します。

JMS では，0 ～ 9 の 10 段階の優先度を定義しています。TP1/Message Queue では，0 ～ MaxPriority の値をとります。MaxPriority は，キューマネジャで定義されています。デフォルト値は 4 です。

パラメタ

defaultPriority

キュー内のメッセージに対する省略時の優先度を指定します。0（最低）～ 9（最高）の範囲の値を指定します。
上記の範囲以外の値を指定した場合，例外 JMSException がスローされます。

例外

JMSException：エラーが発生しました。

● public void setTimeToLive(long timeToLive) throws JMSException

キュー内のメッセージに対する省略時のメッセージの保持時間を設定します。デフォルト値は 0 です。

パラメタ

timeToLive

キュー内のメッセージに対する省略時のメッセージの保持時間（単位：ミリ秒）を指定します。
0 ～（$2^{31} - 1$）× 100 の範囲の値を指定します。ただし，1 ～ 99 の値を指定すると，MQMD 構造体の Expiry フィールドに格納される際に，100 ミリ秒に切り上げられます。
0 を指定した場合，メッセージの保持時間は無制限です。TP1/Message Queue では，MQEI_UNLIMITED（値：-1）に相当します。
0 ～（$2^{31} - 1$）× 100 の範囲以外の値を指定した場合，例外 JMSException がスローされます。

例外

JMSException：エラーが発生しました。

● public Queue getQueue() throws JMSException

該当するキューセンダに関連づけられたキューのオブジェクトを返します。

例外

JMSException：エラーが発生しました。

戻り値：Queue オブジェクト

● public void send(Message message) throws JMSException

キューにメッセージを登録します。

このとき，QueueSender の省略時のメッセージの永続性，メッセージの優先度，およびメッセージの保持時間が使用されます。

QueueSender の生成時にキューが指定されていない場合にこのメソッドを実行すると，例外 JMSException がスローされます。

キューにメッセージを登録する際，Message オブジェクトのプロパティが「JMS メッセージのヘッダとプロパティ」の表 7-6 に示すプロパティ名と一致する場合，対応する MQMD 構造体のフィールドに格納されます。プロパティ名が一致していてプロパティの型が不一致の場合は，表 7-6 に示す型に変換されたあとで，対応する MQMD 構造体のフィールドに格納されます。この変換に失敗すると，例外 MessageFormatException がスローされます。Message オブジェクトのプロパティが表 7-6 に示す名称ではない場合は，プロパティの値は MQMD 構造体のフィールドに格納されません。

パラメタ

message
登録するメッセージを指定します。

例外

JMSException：メッセージの登録に失敗しました。
MessageFormatException：不正なメッセージを指定しました。
InvalidDestinationException：不正なキューに関連づけられた QueueSender のメソッドを実行しました。

● public void send(Message message, int deliveryMode, int priority, long timeToLive) throws JMSException

メッセージの永続性，メッセージの優先度，およびメッセージの保持時間を指定して，キューにメッセージを登録します。

QueueSender の生成時にキューが指定されていない場合にこのメソッドを実行すると，例外 JMSException がスローされます。

キューにメッセージを登録する際，Message オブジェクトのプロパティが「JMS メッセージのヘッダとプロパティ」の表 7-6 に示すプロパティ名と一致する場合，対応する MQMD 構造体のフィールドに格納されます。プロパティ名が一致していてプロパティの型が不一致の場合は，表 7-6 に示す型に変換されたあとで，対応する MQMD 構造体のフィールドに格納されます。この変換に失敗すると，例外 MessageFormatException がスローされます。Message オブジェクトのプロパティが表 7-6 に示す名称ではない場合は，プロパティの値は MQMD 構造体のフィールドに格納されません。

パラメタ

message

登録するメッセージを指定します。

deliveryMode

メッセージの永続性を次の値で指定します。
DeliveryMode.NON_PERSISTENT：非永続
DeliveryMode.PERSISTENT：永続
上記以外の値を指定した場合，例外 JMSException がスローされます。

priority

メッセージの優先度を指定します。0（最低）～ 9（最高）の範囲の値を指定します。
上記の範囲以外の値を指定した場合，例外 JMSException がスローされます。

timeToLive

メッセージの保持時間（単位：ミリ秒）を指定します。
0 ～（$2^{31} - 1$）× 100 の範囲の値を指定します。ただし，1 ～ 99 の値を指定すると，MQMD 構造体の Expiry フィールドに格納される際に，100 ミリ秒に切り上げられます。
0 を指定した場合，メッセージの保持時間は無制限です。TP1/Message Queue では，MQEI_UNLIMITED（値：-1）に相当します。
0 ～（$2^{31} - 1$）× 100 の範囲以外の値を指定した場合，例外 JMSException がスローされます。

例外

JMSException：メッセージの登録に失敗しました。
MessageFormatException：不正なメッセージを指定しました。
InvalidDestinationException：不正なキューに関連づけられた QueueSender のメソッドを実行しました。

● public void send(Queue queue, Message message) throws JMSException

指定したキューにメッセージを登録します。

このとき，QueueSender の省略時のメッセージの永続性，メッセージの優先度，およびメッセージの保持時間が使用されます。

一般に，QueueSender の生成時には登録先のキューが指定されますが，null を指定して，つまりキューを指定しないで QueueSender を生成することもできます。
QueueSender の生成時にキューが指定されていない場合は，このメソッドを実行できます。QueueSender の生成時にキューが指定されている場合にこのメソッドを実行すると，例外 JMSException がスローされます。

キューにメッセージを登録する際，Message オブジェクトのプロパティが「JMS メッセージのヘッダとプロパティ」の表 7-6 に示すプロパティ名と一致する場合，対応する

MQMD 構造体のフィールドに格納されます。プロパティ名が一致していてプロパティの型が不一致の場合は，表 7-6 に示す型に変換されたあとで，対応する MQMD 構造体のフィールドに格納されます。この変換に失敗すると，例外 MessageFormatException がスローされます。Message オブジェクトのプロパティが表 7-6 に示す名称ではない場合は，プロパティの値は MQMD 構造体のフィールドに格納されません。

パラメタ

queue
　登録先のキューを指定します。

message
　登録するメッセージを指定します。

例外

JMSException：メッセージの登録に失敗しました。
MessageFormatException：不正なメッセージを指定しました。
InvalidDestinationException：不正なキューを指定しました。

● public void send(Queue queue, Message message, int deliveryMode, int priority, long timeToLive) throws JMSException

キュー，メッセージの永続性，メッセージの優先度，およびメッセージの保持時間を指定して，キューにメッセージを登録します。

一般に，QueueSender の生成時には登録先のキューが指定されますが，null を指定して，つまりキューを指定しないで QueueSender を生成することもできます。QueueSender の生成時にキューが指定されていない場合は，このメソッドを実行できます。QueueSender の生成時にキューが指定されている場合にこのメソッドを実行すると，例外 JMSException がスローされます。

キューにメッセージを登録する際，Message オブジェクトのプロパティが「JMS メッセージのヘッダとプロパティ」の表 7-6 に示すプロパティ名と一致する場合，対応する MQMD 構造体のフィールドに格納されます。プロパティ名が一致していてプロパティの型が不一致の場合は，表 7-6 に示す型に変換されたあとで，対応する MQMD 構造体のフィールドに格納されます。この変換に失敗すると，例外 MessageFormatException がスローされます。Message オブジェクトのプロパティが表 7-6 に示す名称ではない場合は，プロパティの値は MQMD 構造体のフィールドに格納されません。

パラメタ

queue
　登録先のキューを指定します。

message
　登録するメッセージを指定します。

deliveryMode

メッセージの永続性を次の値で指定します。

DeliveryMode.NON_PERSISTENT：非永続

DeliveryMode.PERSISTENT：永続

上記以外の値を指定した場合，例外 JMSException がスローされます。

priority

メッセージの優先度を指定します。0（最低）～ 9（最高）の範囲の値を指定します。

上記の範囲以外の値を指定した場合，例外 JMSException がスローされます。

timeToLive

メッセージの保持時間（単位：ミリ秒）を指定します。

0 ～（$2^{31} - 1$）× 100 の範囲の値を指定します。ただし，1 ～ 99 の値を指定すると，MQMD 構造体の Expiry フィールドに格納される際に，100 ミリ秒に切り上げられます。

0 を指定した場合，メッセージの保持時間は無制限です。TP1/Message Queue では，MQEI_UNLIMITED（値：-1）に相当します。

0 ～（$2^{31} - 1$）× 100 の範囲以外の値を指定した場合，例外 JMSException がスローされます。

例外

JMSException：メッセージの登録に失敗しました。

MessageFormatException：不正なメッセージを指定しました。

InvalidDestinationException：不正なキューを指定しました。

QueueBrowser インタフェース（JMS）

QueueBrowser インタフェースは，クライアントがキューからメッセージを取り去らないで，参照するために使用します。

形式

```
public interface QueueBrowser
{
  public Queue getQueue() throws JMSException;
  public java.lang.String getMessageSelector() throws JMSException;
  public java.util.Enumeration getEnumeration() throws
JMSException;
  public void close() throws JMSException;
}
```

メソッド

- public Queue getQueue() throws JMSException

該当するキューブラウザに関連づけられたキューのオブジェクトを取得します。

例外

JMSException：キューハンドルが取得できませんでした。

戻り値：Queue オブジェクト

- public java.lang.String getMessageSelector() throws JMSException

メッセージセレクタを返します。

例外

JMSException：エラーが発生しました。

戻り値：メッセージセレクタ

- public java.util.Enumeration getEnumeration() throws JMSException

現在のキューに含まれているメッセージの一覧を読み込んだ順に取得します。メッセージの参照には，このメソッドで取得したメッセージ一覧が使用されます。

例外

JMSException：エラーが発生しました。

戻り値：読み込むメッセージの一覧

- public void close() throws JMSException

TP1/Message Queue が確保した資源を解放するために，資源（キュー）をクローズします。

例外

JMSException：クローズに失敗しました。

QueueConnection インタフェース（JMS）

QueueConnection インタフェースは，アプリケーションが TP1/Message Queue にアクセスするために使用するアプリケーションレベルのハンドルを提供します。

アプリケーションは，QueueConnectionFactory オブジェクトの createQueueConnection メソッドを使用して QueueConnection オブジェクトを取得できます。

QueueConnection の機能を次に示します。

- 受信禁止状態の管理
- コネクションプーリングを利用した QueueSession オブジェクトの生成または取得

形式

```
package:javax.jms

public interface Connection
{
  public void close() throws JMSException;
  public java.lang.String getClientID() throws JMSException;
  public ExceptionListener getExceptionListener() throws
JMSException;
  public ConnectionMetaData getMetaData() throws JMSException;
  public void setClientID(java.lang.String clientID) throws
JMSException;
  public void setExceptionListener(ExceptionListener listener)
throws JMSException;
  public void start() throws JMSException;
  public void stop() throws JMSException;
}

public interface QueueConnection extends javax.jms.Connection
{
  public ConnectionConsumer createConnectionConsumer(Queue queue,
java.lang.String messageSelector, ServerSessionPool sessionPool,
int maxMessages) throws JMSException;
  public QueueSession createQueueSession(boolean transacted, int
acknowledgeMode) throws JMSException;
}
```

メソッド

● public void close() throws JMSException

QueueConnection をクローズします。

例外

JMSException：コネクションのクローズに失敗しました。

● public java.lang.String getClientID() throws JMSException

このメソッドは未サポートです。

例外

JMSException：このメソッドは未サポートです。

● public ExceptionListener getExceptionListener() throws JMSException

このメソッドは未サポートです。

例外

JMSException：このメソッドは未サポートです。

● public ConnectionMetaData getMetaData() throws JMSException

ConnectionMetaData オブジェクトを参照します。

例外

JMSException：QueueConnection がクローズされている状態で，このメソッドが発行されました。

戻り値：ConnectionMetaData オブジェクト

● public void setClientID(java.lang.String clientID) throws JMSException

このメソッドは未サポートです。

パラメタ

clientID

クライアント ID を指定します。

例外

JMSException：このメソッドは未サポートです。
InvlidClientIDException：該当しません。
IllegalStateException：該当しません。

● public void setExceptionListener(ExceptionListener listener) throws JMSException

このメソッドは未サポートです。

パラメタ

listener

ExceptionListener オブジェクトを指定します。

例外

JMSException：このメソッドは未サポートです。

● public void start() throws JMSException

コネクションの受信禁止状態を解除します。

例外

JMSException：コネクションの受信禁止状態の解除に失敗しました。または，QueueConnection がクローズされている状態で，このメソッドが発行されました。

● public void stop() throws JMSException

コネクションを受信禁止状態にします。

コネクションが受信禁止状態になると QueueReciever オブジェクトは停止します。また，すでに受信待ちとなっている QueueReciever オブジェクトの receive メソッドは中断されます。

例外

JMSException：コネクションの受信禁止に失敗しました。または，QueueConnection がクローズされている状態で，このメソッドが発行されました。

● public ConnectionConsumer createConnectionConsumer(Queue queue, java.lang.String messageSelector, ServerSessionPool sessionPool, int maxMessages) throws JMSException

このメソッドは未サポートです。

パラメタ

queue

受信監視する Queue オブジェクトを指定します。

messageSelector

メッセージセレクタを指定します。ただし，指定値は null 固定です。null 以外の文字列を指定しても無視されます。

sessionPool

ServerSessionPool オブジェクトを指定します。

maxMessages

ServerSession オブジェクトが一度に割り当てる最大メッセージ数を指定します。

例外

JMSException：このメソッドは未サポートです。
InvalidDestinationException：該当しません。
InvalidSelectorException：該当しません。

戻り値：ConnectionConsumer オブジェクト

● public QueueSession createQueueSession(boolean transacted, int acknowledgeMode) throws JMSException

コネクションを取得し，QueueSession オブジェクトを生成します。ただし，acknowledgeMode は無視します。

パラメタ

transacted

OpenTP1 でローカルなトランザクション管理をするかどうかを次の値で指定します。

true：ローカルトランザクション管理をします。

false：ローカルトランザクション管理をしません。

acknowledgeMode

QueueSession オブジェクト生成時のモードを次の値で指定します。ただし，指定しても無視されます。

Session.AUTO_ACKNOWLEDGE：AUTO_ACKNOWLEDGE モード

Session.CLIENT_ACKNOWLEDGE：CLIENT_ACKNOWLEDGE モード

Session.DUPS_OK_ACKNOWLEDGE：DUPS_OK_ACKNOWLEDGE モード

例外

JMSException：QueueSession の生成時にエラーが発生しました。または，QueueConnection がクローズされている状態で，このメソッドが発行されました。

戻り値：QueueSession オブジェクト

QueueConnectionFactory インタフェース（JMS）

QueueConnectionFactory インタフェースは，QueueConnection を生成するためのファクトリです。すでに JNDI に登録されている QueueConnectionFactory の JNDI 登録名をアプリケーションが lookup することによって，QueueConnectionFactory オブジェクトを取得できます。

QueueConnectionFactory の機能を次に示します。

- QueueConnection オブジェクトの生成

形式

```
package:javax.jms

public interface QueueConnectionFactory
extends ConnectionFactory, javax.resource.Referenceable
{
  public javax.jms.QueueConnection createQueueConnection() throws
JMSException;
  public javax.jms.QueueConnection
createQueueConnection(java.lang.String userName, java.lang.String
password) throws JMSException;
}
```

メソッド

● public javax.jms.QueueConnection createQueueConnection() throws JMSException

TP1/Message Queue にアクセスするための QueueConnection オブジェクトを生成します。

例外

JMSException：オブジェクトの生成に失敗しました。

JMSSecurityException：該当しません。

戻り値：QueueConnection オブジェクト

● public javax.jms.QueueConnection createQueueConnection(java.lang.String userName, java.lang.String password) throws JMSException

TP1/Message Queue にアクセスするための QueueConnection オブジェクトを生成します。ただし，パラメタで指定されたユーザ名およびパスワードは無視されます。

パラメタ

userName

ユーザ名を指定します。

password

パスワードを指定します。

例外

JMSException：オブジェクトの生成に失敗しました。

JMSSecurityException：該当しません。

戻り値：QueueConnection オブジェクト

QueueSession インタフェース（JMS）

QueueSession インタフェースはアプリケーションからの論理的なコネクションハンドルです。

QueueConnection.createQueueSession で transacted パラメタに true が指定されると，その QueueSession からの操作はトランザクションモードとなります。なお，QueueConnection.createQueueSession での acknowledge パラメタは無視されます。

QueueSession の機能を次に示します。

- ローカルトランザクションの開始，コミット，およびロールバック
- BytesMessage オブジェクトの生成
- Message オブジェクトの生成
- QueueReceiver オブジェクトの生成
- QueueBrowser オブジェクトの生成
- QueueSender オブジェクトの生成

形式

```
public interface Session extends java.lang.Runnable
{
  public static final int AUTO_ACKNOWLEDGE;
  public static final int CLIENT_ACKNOWLEDGE;
  public static final int DUPS_OK_ACKNOWLEDGE;
  public void close() throws JMSException;
  public void commit() throws JMSException;
  public BytesMessage createBytesMessage() throws JMSException;
  public MapMessage createMapMessage() throws JMSException;
  public Message createMessage() throws JMSException;
  public ObjectMessage createObjectMessage() throws JMSException;
  public ObjectMessage createObjectMessage (java.io.Serializable
object) throws JMSException;
  public StreamMessage createStreamMessage() throws JMSException;
  public TextMessage createTextMessage() throws JMSException;
  public TextMessage createTextMessage(java.lang.String text)
throws JMSException;
  public MessageListener getMessageListener() throws JMSException;
  public boolean getTransacted() throws JMSException;
  public void recover() throws JMSException;
  public void rollback() throws JMSException;
  public void run();
  public void setMessageListener(MessageListener listener) throws
JMSException;
}

public interface QueueSession extends Session
{
  public QueueBrowser createBrowser(Queue queue) throws
JMSException;
  public QueueBrowser createBrowser(Queue queue, java.lang.String
messageSelector) throws JMSException;
```

```
  public Queue createQueue(java.lang.String queueName) throws
JMSException;
  public QueueReceiver createReceiver(Queue queue) throws
JMSException;
  public QueueReceiver createReceiver(Queue queue, java.lang.String
messageSelector) throws JMSException;
  public QueueSender createSender(Queue queue) throws JMSException;
  public TemporaryQueue createTemporaryQueue() throws JMSException;
}
```

フィールド

- public static final int AUTO_ACKNOWLEDGE

このフィールドは未サポートです。

- public static final int CLIENT_ACKNOWLEDGE

このフィールドは未サポートです。

- public static final int DUPS_OK_ACKNOWLEDGE

このフィールドは未サポートです。

メソッド

- public void close() throws JMSException

QueueSession をクローズします。

例外

JMSException：QueueSession のクローズに失敗しました。

- public void commit() throws JMSException

ローカルトランザクションをコミットします。

例外

JMSException：コミットに失敗しました。

TransactionRolledBackException：該当しません。

IllegalStateException：メソッドがトランザクション外でコールされました。または，QueueSession がクローズされている状態で，このメソッドが発行されました。

- public BytesMessage createBytesMessage() throws JMSException

BytesMessage オブジェクトを生成します。

例外

JMSException：BytesMessage オブジェクトの生成に失敗しました。または，QueueSession がクローズされている状態で，このメソッドが発行されました。

戻り値：BytesMessage オブジェクト

● public MapMessage createMapMessage() throws JMSException

このメソッドは未サポートです。

例外

JMSException：このメソッドは未サポートです。

● public Message createMessage() throws JMSException

Message オブジェクトを生成します。

例外

JMSException：Message オブジェクトの生成に失敗しました。または，
QueueSession がクローズされている状態で，このメソッドが発行されました。

戻り値：Message オブジェクト

● public ObjectMessage createObjectMessage() throws JMSException

このメソッドは未サポートです。

例外

JMSException：このメソッドは未サポートです。

● public ObjectMessage createObjectMessage(java.io.Serializable object) throws JMSException

このメソッドは未サポートです。

例外

JMSException：このメソッドは未サポートです。

● public StreamMessage createStreamMessage() throws JMSException

このメソッドは未サポートです。

例外

JMSException：このメソッドは未サポートです。

● public TextMessage createTextMessage() throws JMSException

このメソッドは未サポートです。

例外

JMSException：このメソッドは未サポートです。

● public TextMessage createTextMessage(java.lang.String text) throws JMSException

このメソッドは未サポートです。

例外

JMSException：このメソッドは未サポートです。

● public MessageListener getMessageListener() throws JMSException

このメソッドは未サポートです。

例外

JMSException：このメソッドは未サポートです。

● public boolean getTransacted() throws JMSException

QueueSession がトランザクションモードかどうかを取得します。

例外

JMSException：トランザクション状態の取得に失敗しました。または，
QueueSession がクローズされている状態で，このメソッドが発行されました。

戻り値

true：トランザクションモード
false：トランザクションモード以外

● public void recover() throws JMSException

このメソッドは未サポートです。

例外

JMSException：このメソッドは未サポートです。
IllegalStateException：該当しません。

● public void rollback() throws JMSException

ローカルトランザクションをロールバックします。

例外

JMSException：ロールバックに失敗しました。
IllegalStateException：メソッドがトランザクション外でコールされました。または，QueueSession がクローズされている状態で，このメソッドが発行されました。

● public void run()

このメソッドは未サポートです。

● public void setMessageListener(MessageListener listener) throws JMSException

このメソッドは未サポートです。

例外

JMSException：このメソッドは未サポートです。

● public QueueBrowser createBrowser(Queue queue) throws JMSException

キューブラウザを生成します。

パラメタ

queue

アクセス対象の Queue オブジェクトを指定します。
TemporaryQueue オブジェクトを指定してキューブラウザを生成する場合，このメソッドと同一の QueueSession オブジェクトの createTemporaryQueue メソッドによって作成された TemporaryQueue を指定する必要があります。

例外

JMSException：キューブラウザの生成に失敗しました。または，QueueSession がクローズされている状態で，このメソッドが発行されました。
InvalidDestinationException：不正なキューを指定しました。

戻り値：QueueBrowser オブジェクト

● public QueueBrowser createBrowser(Queue queue, java.lang.String messageSelector) throws JMSException

キューブラウザを生成します。

パラメタ

queue

アクセス対象の Queue オブジェクトを指定します。
TemporaryQueue オブジェクトを指定してキューブラウザを生成する場合，このメソッドと同一の QueueSession オブジェクトの createTemporaryQueue メソッドによって作成された TemporaryQueue を指定する必要があります。

messageSelector

メッセージセレクタを指定します。

例外

JMSException：キューブラウザの生成に失敗しました。または，QueueSession がクローズされている状態で，このメソッドが発行されました。
InvalidDestinationException：不正なキューを指定しました。
InvalidSelectorException：不正なメッセージセレクタ構文です。

戻り値：QueueBrowser オブジェクト

● public Queue createQueue(java.lang.String queueName) throws JMSException

このメソッドは未サポートです。Queue オブジェクトは上位クラスで作成されて，JNDI にあらかじめ登録されているためです。

パラメタ

queueName

キュー名称を指定します。

例外

JMSException：このメソッドは未サポートです。

● public QueueReceiver createReceiver(Queue queue) throws JMSException

キューレシーバを生成します。

パラメタ

queue

アクセス対象の Queue オブジェクトを指定します。
TemporaryQueue オブジェクトを指定してキューレシーバを生成する場合，このメソッドと同一の QueueSession オブジェクトの createTemporaryQueue メソッドによって作成された TemporaryQueue を指定する必要があります。

例外

JMSException：キューレシーバの生成に失敗しました。または，QueueSession がクローズされている状態で，このメソッドが発行されました。
InvalidDestinationException：不正なキューを指定しました。

戻り値：QueueReceiver オブジェクト

● public QueueReceiver createReceiver(Queue queue, java.lang.String messageSelector) throws JMSException

キューレシーバを生成します。

パラメタ

queue

アクセス対象の Queue オブジェクトを指定します。
TemporaryQueue オブジェクトを指定してキューレシーバを生成する場合，このメソッドと同一の QueueSession オブジェクトの createTemporaryQueue メソッドによって作成された TemporaryQueue を指定する必要があります。

messageSelector

メッセージセレクタを指定します。

例外

JMSException：キューレシーバの生成に失敗しました。または，QueueSession がクローズされている状態で，このメソッドが発行されました。
InvalidDestinationException：不正なキューを指定しました。
InvalidSelectorException：不正なメッセージセレクタ構文です。

戻り値：QueueReceiver オブジェクト

● public QueueSender createSender(Queue queue) throws JMSException

キューセンダを生成します。

パラメタ

queue

アクセス対象の Queue オブジェクトを指定します。キューが不確定な場合は null を指定してください。
TemporaryQueue オブジェクトを指定してキューセンダを生成する場合，このメソッドと同一の QueueSession オブジェクトの createTemporaryQueue メソッドによって作成された TemporaryQueue を指定する必要があります。

例外

JMSException：キューセンダの生成に失敗しました。または，QueueSession がクローズされている状態で，このメソッドが発行されました。
InvalidDestinationException：不正なキューを指定しました。

戻り値：QueueSender オブジェクト

● public TemporaryQueue createTemporaryQueue() throws JMSException

動的キューを生成します。生成された動的キューは，TemporaryQueue 内の delete メソッドが発行されると削除されます。

createTemporaryQueue メソッドによって作成される動的キューは，QueueSession オブジェクトごとに異なります。

TemporaryQueue を使用するためには，TP1/Message Queue - Access リソースアダプタの Properties 設定の環境変数 ModelQueueName に，モデルキュー名を指定します。詳細については，「2.2　MQC クライアント機能の環境設定」を参照してください。また，サーバ側の TP1/Message Queue でモデルキューを定義する必要があります。このモデルキューの DefinitionType 属性によって，一時的動的キューか永続的動的キューのどちらかが作成されます。作成されるキューの種類ごとに，次に示す点に注意が必要です。

- 一時的動的キューの場合，Cosminexus Component Container のアソシエーション機能が動作することによって，いったんキューが削除されて再作成されます。
  キューが削除されないようにするには，モデルキュー名に永続的動的キューを指定してください。
- 永続的動的キューの場合，要求受信監視タイマなどの無通信時間監視で MQC サーバが待機状態になったとき，削除されないで残ったままとなります。
  クライアントから永続的動的キューを直接削除する方法はないため，サーバで適宜削除してください。

createTemporaryQueue メソッドによって作成される動的キュー名は「JMS*」です（*：ユニークな文字列）。

例外

JMSException：動的キューの生成に失敗しました。または，QueueSession がクローズされている状態で，このメソッドが発行されました。

戻り値：TemporaryQueue オブジェクト

TemporaryQueue インタフェース（JMS）

TemporaryQueue は，QueueConnection がアクティブ状態のときに生成されるユニークな Queue オブジェクトです。

TemporaryQueue は，システムが定義するキューであり，それを生成した QueueConnection だけが利用できます。

一般に，TemporaryQueue はサービス要求の JMSReplyto 宛先として使用されます。

なお，TemporaryQueue オブジェクトに対して getQueueName メソッドを発行すると，モデルキュー名が返されます。

形式

```
public interface TemporaryQueue
extends Queue
{
  public void delete() throws JMSException;
}
```

メソッド

- public void delete() throws JMSException

該当する動的キューを削除します。該当する動的キューを使用中のキューセンダやキューレシーバが残っている場合は，例外 JMSException がスローされます。

例外

JMSException：削除に失敗しました。

Enumeration インタフェース（J2SE）

Enumeration インタフェースは，値を列挙するために使用します。

形式

```
public interface Enumeration
{
  public boolean hasMoreElements();
  public Object nextElement();
}
```

メソッド

● public boolean hasMoreElements()

nextElement メソッドで返却できる要素があるかどうかを判定します。

なお，QueueBrowser の getEnumeration メソッドで返された Enumeration オブジェクトで，このメソッドを発行して false が返された場合，キューからのメッセージの取得に失敗していることがあります。このとき，メッセージ KFCA31309-W が出力されます。また，コネクションが受信禁止状態の場合は，false が返され，メッセージ KFCA31307-W が出力されます。

戻り値

true：オブジェクトに一つ以上の要素が残っています。
false：オブジェクトには要素が一つも残っていません。

● public Object nextElement()

現在位置の要素を返却し，現在位置を次の要素に進めます。getEnumeration メソッド発行後の最初の nextElement メソッド発行では，先頭の要素を返却します。現在位置に返却できる要素がない場合，例外 NoSuchElementException がスローされます。

なお，QueueBrowser の getEnumeration メソッドで返された Enumeration オブジェクトで，このメソッドを発行して例外 NoSuchElementException がスローされた場合，キューからのメッセージの取得に失敗していることがあります。このとき，メッセージ KFCA31310-W が出力されます。また，コネクションが受信禁止状態の場合は，例外 NoSuchElementException がスローされ，メッセージ KFCA31307-W が出力されます。

例外

NoSuchElementException：要素がそれ以上ありません。

戻り値：次の要素となるオブジェクト

MQC インタフェース（JMS）

TP1/Message Queue - Access で指定できる値，およびデフォルト値の定義の一覧を示します。JMS インタフェースでは，プロパティに格納する値としてこれらの変数を使用します。

各変数の詳細については，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。

注意事項

- MQC インタフェースを使用する際には，パッケージ名 jp.co.Hitachi.soft.mqadaptor を指定するか，パッケージ名 jp.co.Hitachi.soft.mqadaptor 中のインタフェース MQC を import 文でインポートする必要があります。
- MQC インタフェースを使用したアプリケーションをコンパイルするときは，classpath に mqcadptdef.jar を追加してください。

変数

- public final static int MQ_CLUSTER_NAME_LENGTH
- public final static int MQ_CORREL_ID_LENGTH
- public final static int MQ_CREATION_DATE_LENGTH
- public final static int MQ_CREATION_TIME_LENGTH
- public final static int MQ_FORMAT_LENGTH
- public final static int MQ_GROUP_ID_LENGTH
- public final static int MQ_MSG_ID_LENGTH
- public final static int MQ_PUT_TIME_LENGTH
- public final static int MQ_PUT_APPL_NAME_LENGTH
- public final static int MQ_PUT_DATE_LENGTH
- public final static int MQ_Q_MGR_NAME_LENGTH
- public final static int MQ_Q_NAME_LENGTH
- public final static int MQ_USER_ID_LENGTH

- public final static int MQAT_UNKNOWN
- public final static int MQAT_NO_CONTEXT
- public final static int MQAT_CICS

- public final static int MQAT_MVS
- public final static int MQAT_IMS
- public final static int MQAT_OS2
- public final static int MQAT_DOS
- public final static int MQAT_AIX
- public final static int MQAT_UNIX
- public final static int MQAT_QMGR
- public final static int MQAT_OS400
- public final static int MQAT_WINDOWS
- public final static int MQAT_CICS_VSE
- public final static int MQAT_VMS
- public final static int MQAT_GUARDIAN
- public final static int MQAT_VOS
- public final static int MQAT_OPEN_TP1
- public final static int MQAT_XDM
- public final static int MQAT_TMS_4V
- public final static int MQAT_DEFAULT
- public final static int MQAT_USER_FIRST
- public final static int MQAT_USER_LAST

- public final static int MQCCSI_EMBEDDED
- public final static int MQCCSI_DEFAULT
- public final static int MQCCSI_Q_MGR
- public final static int MQENC_NATIVE
- public final static int MQENC_INTEGER_MASK
- public final static int MQENC_DECIMAL_MASK
- public final static int MQENC_FLOAT_MASK
- public final static int MQENC_RESERVED_MASK
- public final static int MQENC_INTEGER_UNDEFINED

- public final static int MQENC_INTEGER_NORMAL
- public final static int MQENC_INTEGER_REVERSED
- public final static int MQENC_DECIMAL_UNDEFINED
- public final static int MQENC_DECIMAL_NORMAL
- public final static int MQENC_DECIMAL_REVERSED
- public final static int MQENC_FLOAT_UNDEFINED
- public final static int MQENC_FLOAT_IEEE_NORMAL
- public final static int MQENC_FLOAT_IEEE_REVERSED
- public final static int MQENC_FLOAT_S390
- public final static int MQFB_NONE
- public final static int MQFB_SYSTEM_FIRST
- public final static int MQFB_EXPIRATION
- public final static int MQFB_COA
- public final static int MQFB_COD
- public final static int MQFB_QUIT
- public final static int MQFB_APPL_CANNOT_BE_STARTED
- public final static int MQFB_TM_ERROR
- public final static int MQFB_APPL_TYPE_ERROR
- public final static int MQFB_STOPPED_BY_MSG_EXIT
- public final static int MQFB_XMIT_Q_MSG_ERROR
- public final static int MQFB_SYSTEM_LAST
- public final static int MQFB_APPL_FIRST
- public final static int MQFB_APPL_LAST
- public final static int MQFB_NOT_A_REPOSITORY_MSG
- public final static int MQFB_PAN
- public final static int MQFB_NAN
- public final static String MQFMT_NONE
- public final static String MQFMT_ADMIN
- public final static String MQFMT_DEAD_LETTER_HEADER

- public final static String MQFMT_EVENT
- public final static String MQFMT_PCF
- public final static String MQFMT_STRING
- public final static String MQFMT_TRIGGER
- public final static String MQFMT_XMIT_Q_HEADER
- public final static String MQFMT_DIST_HEADER
- public final static String MQFMT_MD_EXTENSION
- public final static String MQFMT_REF_MSG_HEADER
- public final static byte[] MQGI_NONE
- public final static int MQMT_SYSTEM_FIRST
- public final static int MQMT_REQUEST
- public final static int MQMT_REPLY
- public final static int MQMT_DATAGRAM
- public final static int MQMT_REPORT
- public final static int MQMT_SYSTEM_LAST
- public final static int MQMT_APPL_FIRST
- public final static int MQMT_APPL_LAST
- public final static int MQRO_EXCEPTION
- public final static int MQRO_EXCEPTION_WITH_DATA
- public final static int MQRO_EXCEPTION_WITH_FULL_DATA
- public final static int MQRO_EXPIRATION
- public final static int MQRO_EXPIRATION_WITH_DATA
- public final static int MQRO_EXPIRATION_WITH_FULL_DATA
- public final static int MQRO_COA
- public final static int MQRO_COA_WITH_DATA
- public final static int MQRO_COA_WITH_FULL_DATA
- public final static int MQRO_COD
- public final static int MQRO_COD_WITH_DATA
- public final static int MQRO_COD_WITH_FULL_DATA

- public final static int MQRO_COPY_MSG_ID_TO_CORREL_ID
- public final static int MQRO_PASS_CORREL_ID
- public final static int MQRO_NEW_MSG_ID
- public final static int MQRO_PASS_MSG_ID
- public final static int MQRO_DEAD_LETTER_Q
- public final static int MQRO_DISCARD_MSG
- public final static int MQRO_NONE
- public final static int MQRO_REJECT_UNSUP_MASK
- public final static int MQRO_ACCEPT_UNSUP_MASK
- public final static int MQRO_ACCEPT_UNSUP_IF_XMIT_MASK
- public final static int MQRO_PAN
- public final static int MQRO_NAN

JMS インタフェースのサンプルアプリケーション

JMS インタフェースのサンプルアプリケーションの処理の流れと，サンプルアプリケーションを Cosminexus Component Container で実行する手順について説明します。

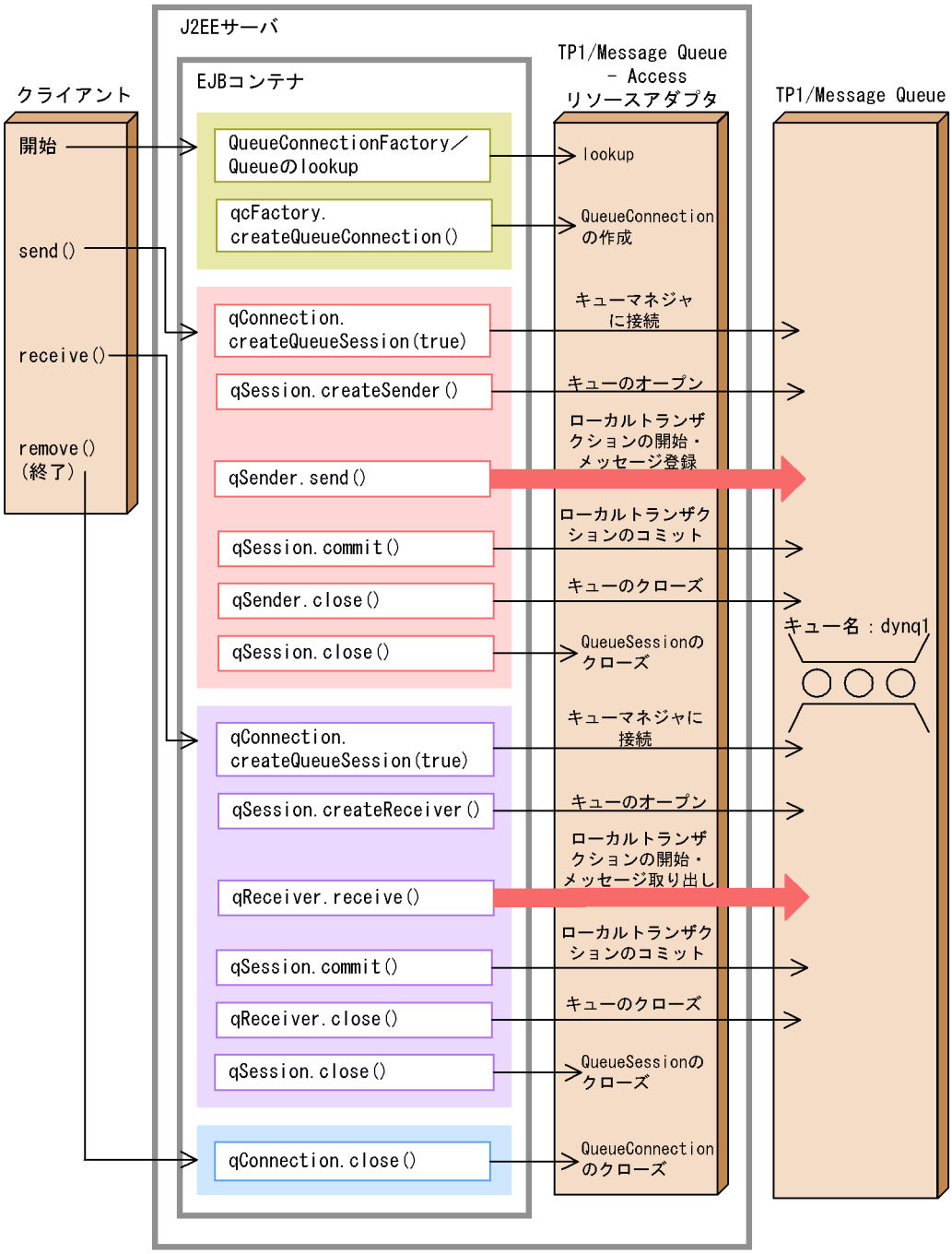
JMS インタフェースのサンプルアプリケーションとして，TP1/Message Queue - Access のインストール先の examples ディレクトリ下の jms ディレクトリに SessionBean1 と SessionBean2 の 2 種類が格納されます。

- SessionBean1 は，JMS インタフェースを使用してメッセージの登録と取り出しを実行します。
- SessionBean2 は，SessionBean1 の処理に加えて，登録するメッセージの JMS プロパティに MQC インタフェースの値を設定したり，取り出したメッセージの JMS プロパティから値を取得したりします。

サンプルアプリケーションの処理の流れ

サンプルアプリケーションの処理の流れを次の図に示します。

図 7-4　処理の流れ（JMS インタフェースの場合）



Cosminexus Component Container での実行手順

J2EE サーバの名称は「MyServer」，J2EE アプリケーションの Display name は

「MQAccess」としてください。また，登録先のキューにメッセージが残っていないことを確認してください。

このサンプルアプリケーションでは Cosminexus Component Container が起動するネームサーバのポート番号をデフォルト値（900）としています。変更している場合は，次のファイルに定義されている環境変数 PROVIDER_URL を修正してください。

- Windows の場合：deployApp.bat，unDeployApp.bat，および testClient.bat
- UNIX の場合：deployApp，unDeployApp，および testClient

Windows の場合の実行手順

1. サンプルアプリケーションの格納ディレクトリ（%MQCDIR%¥examples¥jms¥SessionBean1 または %MQCDIR%¥examples¥jms¥SessionBean2）まで移動します。
2. compileBean.bat，compileClient.bat，deployApp.bat，unDeployApp.bat，および testClient.bat で定義されている環境変数 COSMI_HOME を Cosminexus のインストール先ディレクトリに修正します。
3. SessionBean2 の場合だけ，compileBean.bat で定義されている環境変数 MQACS_HOME を TP1/Message Queue - Access のインストール先ディレクトリに修正します。
4. compileBean.bat → compileClient.bat の順に実行します。
   同じディレクトリに jmssample1.jar または jmssample2.jar という EJB-JAR ファイルが作成されます。また，同じディレクトリに jmssample1.ear または jmssample2.ear というアプリケーションファイルが作成されます。
5. Cosminexus Component Container で，TP1/Message Queue - Access リソースアダプタをインポートします。
6. TP1/Message Queue - Access リソースアダプタの <config-property> の QueueConfigFileName に，キュー定義ファイル QueueConfig を完全パスで指定します。また，その他の環境変数も適宜設定します。環境変数の詳細については，「2.2 MQC クライアント機能の環境設定」を参照してください。
7. 6. で指定したキュー定義ファイル QueueConfig の Queue.1.DisplayName，Queue.1.QueueName，および Queue.1.QueueManagerName の記述が正しいことを確認します。
8. TP1/Message Queue - Access リソースアダプタを追加し，開始します。
9. deployApp.bat を実行し，J2EE アプリケーション MQAccess を開始します。
10. testClient.bat を実行します。
11. unDeployApp.bat を実行し，J2EE アプリケーション MQAccess を停止，および削除します。

注意事項

4. で生成される EJB-JAR ファイルを使って，GUI サーバで操作することもできます。この場合，testClient.bat の前に次のコマンドを実行して RMI-IIOP スタブ（stubs.jar）を取得してください。
cjgetstubsjar MyServer
-nameserver corbaname::localhost:****
-name MQAccess -d .
-nameserver の **** には Cosminexus Component Container が起動するネームサーバのポート番号を指定してください（デフォルト値は 900）。ただし，cjgetstubsjar コマンドは GUI サーバの起動中は実行できないため，GUI サーバを停止する必要があります。

UNIX の場合の実行手順

1. サンプルアプリケーションの格納ディレクトリ（$MQCDIR/examples/jms/SessionBean1 または $MQCDIR/examples /jms/SessionBean2）まで移動します。
2. compileBean，compileClient，および testClient.bat で定義されている環境変数 COSMI_HOME を Cosminexus のインストール先ディレクトリに修正します。
3. SessionBean2 の場合だけ，compileBean で定義されている環境変数 MQACS_HOME を TP1/Message Queue - Access のインストール先ディレクトリに修正します。
4. compileBean → compileClient の順に実行します。
同じディレクトリに jmssample1.jar または jmssample2.jar という EJB-JAR ファイルが作成されます。
5. Cosminexus Component Container で，TP1/Message Queue - Access リソースアダプタをインポートします。
6. TP1/Message Queue - Access リソースアダプタのプロパティ Configurations の QueueConfigFileName に，キュー定義ファイル QueueConfig を完全パスで指定します。また，その他の環境変数も適宜設定します。環境変数の詳細については，「2.2 MQC クライアント機能の環境設定」を参照してください。
7. 6. で指定したキュー定義ファイル QueueConfig の Queue.1.DisplayName，Queue.1.QueueName，および Queue.1.QueueManagerName の記述が正しいことを確認します。
8. TP1/Message Queue - Access リソースアダプタを追加し，開始します。
9. deployApp を実行し，J2EE アプリケーション MQAccess を開始します。
10. testClient を実行します。
11. unDeployApp を実行し，J2EE アプリケーション MQAccess を停止，および削除します。

注意事項

4. で生成される EJB-JAR ファイルを使って，GUI サーバで操作することもできます。この場合，testClient の前に次のコマンドを実行して RMI-IIOP スタブ（stubs.jar）を取得してください。

cjgetstubsjar MyServer
-nameserver corbaname::localhost:****
-name MQAccess -d .

-nameserver の **** には Cosminexus Component Container が起動するネームサーバのポート番号を指定してください（デフォルト値は 900）。ただし，cjgetstubsjar コマンドは GUI サーバの起動中は実行できないため，GUI サーバを停止する必要があります。

JMS インタフェースのサンプルコーディング

コーディング例

```
// All Rights Reserved. Copyright (C) 2003, Hitachi, Ltd.
/*********************************************************
**  JMSSample1EJB.java
**
*********************************************************/
import javax.jms.QueueConnectionFactory;
import javax.jms.QueueConnection;
import javax.jms.QueueSession;
import javax.jms.QueueSender;
import javax.jms.QueueReceiver;
import javax.jms.Queue;
import javax.jms.Session;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import javax.ejb.EJBException;
import javax.ejb.CreateException;
import java.rmi.RemoteException;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class JMSSample1EJB implements SessionBean {

  private SessionContext sContext = null;
  private String clientName = null;
  private QueueConnection qConnection = null;
  private Queue queue = null;

  private static final String queueLookupName =
                              "java:comp/env/jms/queue";
  private static final String cfLookupName =
                              "java:comp/env/jms/qcf";

/*********************************************************
**  name = send()
**  func = initialize a queue sender and send a message
**       (1)createQueueSession(create a queue session)
**       (2)createSender(create a queue sender)
**       (3)send(send the message)
**       (4)commit(commit local transaction)
**       (5)close(close the queue sender)
**       (6)close(close the queue session)
*********************************************************/
  public void send() {

    QueueSession qSession = null;
    QueueSender qSender = null;
    BytesMessage putMessage = null;
```

```
try {
  // create a queue session
  qSession =
    qConnection.createQueueSession(true,
                              Session.AUTO_ACKNOWLEDGE);

  // create a queue sender
  qSender = qSession.createSender(queue);

  // create a message object
  putMessage = qSession.createBytesMessage();

  // UTF format message data
  putMessage.
         writeUTF("******** sample put data " +
                        this.clientName + " ********");

  // send the message
  qSender.send(putMessage);

  // commit local transaction
  qSession.commit();

  // close the queue sender
  qSender.close();

} catch (JMSException ex) {
  // JMS error
  System.out.println
    ("An error occurred in send(): " + ex.getMessage()
               + " error code = " + ex.getErrorCode());
  ex.printStackTrace();
  throw new EJBException(ex.getMessage());
} catch(Exception e) {
  // other exception
  System.err.println("An exception was thrown: "
                                      + e.getMessage());
  e.printStackTrace();
  throw new EJBException(e.getMessage());
} finally {
  if(qSession != null) {
    try {
      // close the queue session
      qSession.close();

    } catch (JMSException ex) {
      // JMS error
      System.out.println
        ("An error occurred in send(): "
                   + ex.getMessage() + " error code = "
                   + ex.getErrorCode());
      ex.printStackTrace();
      throw new EJBException(ex.getMessage());
    } catch(Exception e) {
      // other exception
      System.err.println("An exception was thrown: "
                                      + e.getMessage());
      e.printStackTrace();
```

```
            throw new EJBException(e.getMessage());
          }
        }
      }
    }

  /*********************************************************
  **  name = receive()
  **  func = initialize a queue receiver and
  **                                        receive a message
  **      (1)start(start the queue connection)
  **      (2)createQueueSession(create a queue session)
  **      (3)createReceiver(create a queue receiver)
  **      (4)receive(receive a message)
  **      (5)commit(commit local transaction)
  **      (6)close(close the queue receiver)
  **      (7)close(close the queue session)
  *********************************************************/
    public void receive() {

      QueueSession qSession = null;
      QueueReceiver qReceiver = null;
      BytesMessage getMessage = null;

      try {
        // start the queue connection
        qConnection.start();

        // create a queue session
        qSession =
          qConnection.createQueueSession(true,
                                  Session.AUTO_ACKNOWLEDGE);

        // create a queue receiver
        qReceiver = qSession.createReceiver(queue);

        // receive a message
        getMessage = (BytesMessage)qReceiver.receive(1000);

        // commit local transaction
        qSession.commit();

        if(getMessage != null) {
          // display the get message
          String msgText = getMessage.readUTF();
          System.out.println("The message is: " + msgText);
        }

        // close the queue receiver
        qReceiver.close();

      } catch (JMSException ex) {
        // JMS error
        System.out.println
          ("An error occurred in receive(): "
                        + ex.getMessage() + " error code = "
                        + ex.getErrorCode());
        ex.printStackTrace();
```

```
      throw new EJBException(ex.getMessage());
    } catch(Exception e) {
      // other exception
      System.err.println("An exception was thrown: "
                                          + e.getMessage());
      e.printStackTrace();
      throw new EJBException(e.getMessage());
    } finally {
      if(qSession != null) {
        try {
          // close the queue session
          qSession.close();

        } catch (JMSException ex) {
          // JMS error
          System.out.println
            ("An error occurred in receive(): "
                      + ex.getMessage() + " error code = "
                      + ex.getErrorCode());
          ex.printStackTrace();
          throw new EJBException(ex.getMessage());
        } catch(Exception e) {
          // other exception
          System.err.println("An exception was thrown: "
                                          + e.getMessage());
          e.printStackTrace();
          throw new EJBException(e.getMessage());
        }
      }
    }
  }

/************************************************************
**  name = setSessionContext()
**  func = set a session context
************************************************************/
  public void setSessionContext(SessionContext sc) {

    this.sContext = sc;

  }

/************************************************************
**  name = ejbCreate()
**  func =
**      (1)lookup(look up a connection factory and
**                                          a queue object)
**      (2)createQueueConnection(create a queue connection)
************************************************************/
  public void ejbCreate(String name)
                                  throws CreateException {

    Context ic = null;
    QueueConnectionFactory qcFactory = null;
    this.clientName = name;

    try {
      // create a context
```

```
        ic = new InitialContext();

        // looking up ConnectionFactory
        qcFactory
           = (QueueConnectionFactory)ic.lookup(cfLookupName);

        // look up a queue object
        queue = (Queue)ic.lookup(queueLookupName);

        // create a queue connection
        qConnection = qcFactory.createQueueConnection();

      } catch (JMSException ex) {
        // JMS error
        System.out.println
          ("An error occurred in ejbCreate(): "
                        + ex.getMessage() + " error code = "
                        + ex.getErrorCode());
        ex.printStackTrace();
        throw new CreateException(ex.getMessage());
      } catch (NamingException nex) {
        System.err.println("A NamingException was thrown: "
                                          + nex.getMessage());
        nex.printStackTrace();
        throw new CreateException(nex.getMessage());
      } catch (Exception e) {
        System.err.println("An exception was thrown: "
                                            + e.getMessage());
        e.printStackTrace();
        throw new CreateException(e.getMessage());
      }

    }

  /*********************************************************
  **  name = ejbRemove()
  **  func =
  **      (1)close(close the queue connection)
  *********************************************************/
    public void ejbRemove() {

      try {

        if(qConnection != null) {
          // close the queue connection
          qConnection.close();
        }

      } catch (JMSException ex) {
        // JMS error
        System.out.println
          ("An error occurred in ejbRemove(): "
                        + ex.getMessage() + " error code = "
                        + ex.getErrorCode());
        ex.printStackTrace();
        throw new EJBException(ex.getMessage());
      } catch(Exception e) {
        // other exception
```

```
      System.err.println("An exception was thrown: "
                                          + e.getMessage());
      e.printStackTrace();
      throw new EJBException(e.getMessage());
    }

  }

  public void ejbActivate() {}
  public void ejbPassivate() {}
}
```

コーディング例（MQC インタフェース使用時）

```
// All Rights Reserved. Copyright (C) 2003, Hitachi, Ltd.
/*********************************************************
**  JMSSample2EJB.java
**
*********************************************************/
import javax.jms.QueueConnectionFactory;
import javax.jms.QueueConnection;
import javax.jms.QueueSession;
import javax.jms.QueueSender;
import javax.jms.QueueReceiver;
import javax.jms.Queue;
import javax.jms.Session;
import javax.jms.BytesMessage;
import javax.jms.JMSException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import javax.ejb.EJBException;
import javax.ejb.CreateException;
import java.rmi.RemoteException;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import jp.co.Hitachi.soft.mqadaptor.MQC;

public class JMSSample2EJB implements SessionBean {

  private SessionContext sContext = null;
  private String clientName = null;
  private QueueConnection qConnection = null;
  private Queue queue = null;

  private static final String queueLookupName =
                                "java:comp/env/jms/queue";
  private static final String cfLookupName =
                                "java:comp/env/jms/qcf";

/*********************************************************
**  name = send()
**  func = initialize a queue sender and send a message
**       (1)createQueueSession(create a queue session)
**       (2)createSender(create a queue sender)
**       (3)send(send the message)
**       (4)commit(commit local transaction)
```

```
**      (5)close(close the queue sender)
**      (6)close(close the queue session)
*********************************************************/
  public void send() {

    QueueSession qSession = null;
    QueueSender qSender = null;
    BytesMessage putMessage = null;

    try {
      // create a queue session
      qSession =
        qConnection.createQueueSession(true,
                                  Session.AUTO_ACKNOWLEDGE);

      // create a queue sender
      qSender = qSession.createSender(queue);

      // create a message object
      putMessage = qSession.createBytesMessage();

      // UTF format message data
      putMessage.
             writeUTF("******** sample put data " +
                         this.clientName + " ********");

      // set a property
      putMessage.setStringProperty(
                    "JMS_HITACHI_Format",MQC.MQFMT_STRING);

      // send the message
      qSender.send(putMessage);

      // commit local transaction
      qSession.commit();

      // close the queue sender
      qSender.close();

    } catch (JMSException ex) {
      // JMS error
      System.out.println
        ("An error occurred in send(): " + ex.getMessage()
                    + " error code = " + ex.getErrorCode());
      ex.printStackTrace();
      throw new EJBException(ex.getMessage());
    } catch(Exception e) {
      // other exception
      System.err.println("An exception was thrown: "
                                          + e.getMessage());
      e.printStackTrace();
      throw new EJBException(e.getMessage());
    } finally {
      if(qSession != null) {
        try {
          // close the queue session
          qSession.close();
```

```
        } catch (JMSException ex) {
          // JMS error
          System.out.println
            ("An error occurred in send(): "
                      + ex.getMessage() + " error code = "
                      + ex.getErrorCode());
          ex.printStackTrace();
          throw new EJBException(ex.getMessage());
        } catch(Exception e) {
          // other exception
          System.err.println("An exception was thrown: "
                                          + e.getMessage());
          e.printStackTrace();
          throw new EJBException(e.getMessage());
        }
      }
    }
  }

/*********************************************************
**  name = receive()
**  func = initialize a queue receiver and
**                                        receive a message
**      (1)start(start the queue connection)
**      (2)createQueueSession(create a queue session)
**      (3)createReceiver(create a queue receiver)
**      (4)receive(receive a message)
**      (5)commit(commit local transaction)
**      (6)close(close the queue receiver)
**      (7)close(close the queue session)
*********************************************************/
  public void receive() {

    QueueSession qSession = null;
    QueueReceiver qReceiver = null;
    BytesMessage getMessage = null;

    try {
      // start the queue connection
      qConnection.start();

      // create a queue session
      qSession =
        qConnection.createQueueSession(true,
                                  Session.AUTO_ACKNOWLEDGE);

      // create a queue receiver
      qReceiver = qSession.createReceiver(queue);

      // receive a message
      getMessage = (BytesMessage)qReceiver.receive(1000);

      // commit local transaction
      qSession.commit();

      if(getMessage != null) {
        // display the get message
        String msgText = getMessage.readUTF();
```

```
        String msgFormat = getMessage.
                    getStringProperty("JMS_HITACHI_Format");
        System.out.println("The message is: " + msgText);
        System.out.println(
                     "The message format is: " + msgFormat);
      }

      // close the queue receiver
      qReceiver.close();

    } catch (JMSException ex) {
      // JMS error
      System.out.println
        ("An error occurred in receive(): "
                      + ex.getMessage() + " error code = "
                      + ex.getErrorCode());
      ex.printStackTrace();
      throw new EJBException(ex.getMessage());
    } catch(Exception e) {
      // other exception
      System.err.println("An exception was thrown: "
                                          + e.getMessage());
      e.printStackTrace();
      throw new EJBException(e.getMessage());
    } finally {
      if(qSession != null) {
        try {
          // close the queue session
          qSession.close();

        } catch (JMSException ex) {
          // JMS error
          System.out.println
            ("An error occurred in receive(): "
                      + ex.getMessage() + " error code = "
                      + ex.getErrorCode());
          ex.printStackTrace();
          throw new EJBException(ex.getMessage());
        } catch(Exception e) {
          // other exception
          System.err.println("An exception was thrown: "
                                          + e.getMessage());
          e.printStackTrace();
          throw new EJBException(e.getMessage());
        }
      }
    }
  }

/**********************************************************
**  name = setSessionContext()
**  func = set a session context
**********************************************************/
  public void setSessionContext(SessionContext sc) {

    this.sContext = sc;

  }
```

```
/*********************************************************
**  name = ejbCreate()
**  func =
**      (1)lookup(look up a connection factory and
**                                          a queue object)
**      (2)createQueueConnection(create a queue connection)
*********************************************************/
  public void ejbCreate(String name)
                                  throws CreateException {

    Context ic = null;
    QueueConnectionFactory qcFactory = null;
    this.clientName = name;

    try {
      // create a context
      ic = new InitialContext();

      // looking up ConnectionFactory
      qcFactory
         = (QueueConnectionFactory)ic.lookup(cfLookupName);

      // look up a queue object
      queue = (Queue)ic.lookup(queueLookupName);

      // create a queue connection
      qConnection = qcFactory.createQueueConnection();

    } catch (JMSException ex) {
      // JMS error
      System.out.println
        ("An error occurred in ejbCreate(): "
                      + ex.getMessage() + " error code = "
                      + ex.getErrorCode());
      ex.printStackTrace();
      throw new CreateException(ex.getMessage());
    } catch (NamingException nex) {
      System.err.println("A NamingException was thrown: "
                                      + nex.getMessage());
      nex.printStackTrace();
      throw new CreateException(nex.getMessage());
    } catch (Exception e) {
      System.err.println("An exception was thrown: "
                                        + e.getMessage());
      e.printStackTrace();
      throw new CreateException(e.getMessage());
    }

  }

/*********************************************************
**  name = ejbRemove()
**  func =
**      (1)close(close the queue connection)
*********************************************************/
  public void ejbRemove() {
```

```
        try {

          if(qConnection != null) {
            // close the queue connection
            qConnection.close();
          }

        } catch (JMSException ex) {
          // JMS error
          System.out.println
            ("An error occurred in ejbRemove(): "
                          + ex.getMessage() + " error code = "
                          + ex.getErrorCode());
          ex.printStackTrace();
          throw new EJBException(ex.getMessage());
        } catch(Exception e) {
          // other exception
          System.err.println("An exception was thrown: "
                                              + e.getMessage());
          e.printStackTrace();
          throw new EJBException(e.getMessage());
        }

      }

      public void ejbActivate() {}
      public void ejbPassivate() {}
    }
```

8 メッセージの一覧

この章では，MQC クライアント機能が出力するメッセージについて説明します。MQC サーバ機能が出力するメッセージについては，マニュアル「TP1/Message Queue メッセージ」を参照してください。

8.1 メッセージの形式

8.2 メッセージ一覧

8.1 メッセージの形式

8.1.1 出力形式

MQC クライアント機能で出力されるメッセージの形式を示します。

```
KFCAnnnnn-X     YY....YY
```

- KFCAnnnnn-X：メッセージ ID（半角英数字 11 文字）
- YY....YY：メッセージテキスト（最大 222 バイト）

8.1.2 記述形式

メッセージの記述形式を次に示します。

KFCAnnnnn-X

メッセージテキスト（Y）

メッセージの意味を説明します。

（S）システムがメッセージを出力したあとにする主な処理を示します。

（O）メッセージ確認時のオペレータの処置を示します。

〔対策〕メッセージ確認時の OpenTP1 管理者の処置を示します。

注

メッセージ中の**（O）**または**〔対策〕**で，「保守員に連絡してください。」とは，当社社員，または当社営業担当部署に連絡することを示します。

8.1.3 メッセージ ID の記号の説明

メッセージ ID の記号の意味は次のとおりです。

KFCA：OpenTP1，または TP1/Message Queue - Access のメッセージであることを示します。

nnnnn：メッセージの通し番号を示します。

X：メッセージの種類を示します。

記号の種類と意味を次の表に示します。

表 8-1 メッセージの種類

種類	意味
E	• 各ライブラリ，コマンド，サーバの機能が働かない障害が起きたことを示しています。 • 定義誤り，コマンドのオペランド指定誤りによって，動作できないことを示しています。
W	• 各ライブラリ，コマンド，サーバからのメモリの使用状況についての警告を示しています。 • 定義誤り，コマンドのオペランド指定誤りはありましたが，値を仮定して動作を実行することを示しています。
I	• 上記の E，W に該当しないメッセージで，動作の報告を示しています。
R	• 出力メッセージに対するユーザからの応答を待っていることを示しています。

(Y)：メッセージの出力先種別を示します。

出力先種別を次の表に示します。

表 8-2 メッセージの出力先種別

種別	出力先
C	コンソール
E	標準エラー出力
S	標準出力
L	メッセージログファイル
R	エラーログファイル（syslog またはイベントログを含む） JMS インタフェース関連のメッセージ（KFCA31300 ～ KFCA31399）は，Cosminexus で指定する LogWriter に出力されます。
O	オンライン端末

8.2 メッセージ一覧

MQC クライアント機能が出力するメッセージを次に示します。MQC サーバ機能が出力するメッセージについては，マニュアル「TP1/Message Queue メッセージ」を参照してください。

KFCA30950-E

the environment variable is not set. environment variable name=aa....aa（R）

aa....aa：環境変数名

(S)API 実行がエラーリターンします。

(O) 環境変数が設定されているか確認して，再度実行してください。

KFCA30951-E

the value specified to the environment variable is invalid. environment variable name =aa....aa(R)

aa....aa：環境変数名

(S)API 実行がエラーリターンします。

(O) 環境変数に指定した値を確認して，再度実行してください。

KFCA30952-E

A format error occurred in a connection destination file. file = aa....aa

接続先情報定義ファイルのフォーマットエラーを検出しました。

aa....aa：接続先情報定義ファイル名

(S)API 実行がエラーリターンします。

(O) キューマネジャ構成定義が正しく設定されているかどうかを確認してください。また，設定した環境変数が正しく設定されているかどうかを確認してください。メッセージ KFCA30953-W，またはメッセージ KFCA30954-W が出力されている場合，メッセージの指示に従って処置してください。

KFCA30953-W

the environment variable is not set. file = aa....aa, environment variable name = bb....bb, line = cc....cc

接続先情報定義ファイルの必須の環境変数が設定されていません。

aa....aa：接続先情報定義ファイル名

bb....bb：環境変数名

cc....cc：行番号

(S) 処理を続行します。

(O) キューマネジャ構成定義の先頭行が cc....cc に表示されるため，該当するキューマネジャ構成定義に，環境変数を指定しているかどうかを確認してください。

KFCA30954-W

the value specified to the environment variable is invalid. file = aa....aa, environment variable name = bb....bb, line = cc....cc

接続先情報定義ファイルの環境変数で指定した値に誤りがあります。

aa....aa：接続先情報定義ファイル名

bb....bb：環境変数名

cc....cc：行番号

(S) 処理を続行します。

(O) 行番号で表示された，環境変数に指定した値を確認してください。

KFCA30960-E

an error occurred in the TCP/IP interface. Own(aa.....aa, bb.....bb) Partner(cc.....cc, dd.....dd) (ee.....ee, ff.....ff, gg.....gg, hhh)（R）

aa.....aa：自システムの IP アドレス

bb.....bb：自システムのポート番号

cc.....cc：相手システムの IP アドレス

dd.....dd：相手システムのポート番号

ee.....ee：自システムのプロセス ID

ff.....ff：自システムのスレッド ID

gg.....gg：ソケット関数名

hhh：TCP/IP の障害エラー番号

障害エラー番号がない場合は *** を出力します。

(S)API 実行がエラーリターンします。

(O) サーバから TCP/IP コネクションが切断された場合や回線状態が不安定な場合以外に出力されたときは，OpenTP1 管理者に連絡してください。

〔対策〕OS のマニュアルから，ソケット関数名に示す関数のエラーコードを調査してください。障害エラー番号に *** が出力された場合は，相手システムの状態を確認してください。ソケット関数名が connect であった場合は，障害エラー番号に対応した次の確認をしてください。

3：非同期型 connect 関数の完了待ちでタイムアウトを検知しています。相手システムの状態を確認してください。

-6960：TP1/Message Queue - Access の障害対策資料を採取し，保守員に連絡してください。

KFCA30961-E

the invalid data was recieved. Own(aa.....aa, bb.....bb) Partner(cc.....cc, dd.....dd) (ee.....ee, ff.....ff, ggg)（R）

aa.....aa：自システムの IP アドレス

bb.....bb：自システムのポート番号

cc.....cc：相手システムの IP アドレス

dd.....dd：相手システムのポート番号

ee.....ee：自システムのプロセス ID

ff.....ff：自システムのスレッド ID

ggg：理由コード

(S)API 実行がエラーリターンします。

(O) OpenTP1 管理者に連絡してください。

〔対策〕保守員に連絡してください。

KFCA30962-E

the protocol error occurred. IP address = aa.....aa port number= bbb maintenance information= ccc (R)

aa.....aa：相手システムの IP アドレス

bbb：相手システムのポート番号

ccc：保守情報

(S)API 実行がエラーリターンします。

(O) OpenTP1 管理者に連絡してください。

〔対策〕保守員に連絡してください。

KFCA30963-E

a time-out occurred while waiting to recieve the next segment. Own(aa.....aa, bb.....bb) Partner(cc.....cc, dd.....dd) (ee.....ee, ff.....ff)（R）

aa.....aa：自システムの IP アドレス

bb.....bb：自システムのポート番号

cc.....cc：相手システムの IP アドレス

dd.....dd：相手システムのポート番号

ee.....ee：自システムのプロセス ID

ff.....ff：自システムのスレッド ID

(S)API 実行がエラーリターンします。

(O) 環境変数 DCMQCTIMESEG の MQC リスナサーバまたは MQC ゲートウェイサーバからの継続セグメント受信監視タイマ値を見直してください。

KFCA30964-E

a time-out occurred while waiting to recieve a message. Own(aa.....aa, bb.....bb) Partner(cc.....cc, dd.....dd) (ee.....ee, ff.....ff)（R）

aa.....aa：自システムの IP アドレス

bb.....bb：自システムのポート番号

cc.....cc：相手システムの IP アドレス

dd.....dd：相手システムのポート番号

ee.....ee：自システムのプロセス ID

ff.....ff：自システムのスレッド ID

(S)API 実行がエラーリターンします。

(O) OpenTP1 管理者に連絡してください。

〔対策〕通信相手のシステム状態を確認し，環境変数 DCMQCTIMEREQ の MQC リスナサーバまたは MQC ゲートウェイサーバからの結果応答受信監視タイマ値を見直してください。

KFCA30965-E

the host information is not set in aa....aa. host information = bb....bb（R）

aa....aa：情報ファイル名

bb....bb：指定ホスト名

(S) 処理を終了します。

(O) OpenTP1 管理者に連絡してください。

〔対策〕ホスト名およびホスト名ファイルの指定に誤りがないか確認してください。

KFCA30966-E

the service information is not set in aa....aa. service information = bb...bb（R）

aa....aa：情報ファイル名

bb....bb：指定サービス名

(S) 処理を終了します。

(O) OpenTP1 管理者に連絡してください。

〔対策〕サービス名およびサービス名ファイルの指定に誤りがないか確認してください。

KFCA30967-E

a local memory shortage occurred. required memory:aaa（R）

aaa：メモリサイズ（単位：バイト）

(S) 処理を終了します。

(O) OpenTP1 管理者に連絡してください。

〔対策〕十分なメモリ容量を確保してください。

KFCA30968-W

The API tracing function will now be reduced and processing will continue.（R）

(S) 機能を縮退して処理を続行します。

(O) OpenTP1 管理者に連絡してください。

〔対策〕直前に出力されたメッセージを確認してください。

KFCA30969-E

an error occurred in the MQC transaction control. information = aaa（R）

aaa：保守情報

(S) 処理を終了します。

(O) OpenTP1 管理者に連絡してください。

〔対策〕環境変数に DCMQCEXPTRN=Y を設定してください。

KFCA30970-E

メモリ不足が発生しました。(E)

(S) 処理を終了します。

(O) OpenTP1 管理者に連絡してください。

〔対策〕十分なメモリを確保し，再度実行してください。

KFCA30971-E

オプションフラグが不正です。(E)

(S) 処理を終了します。

(O) 指定したコマンド引数およびオプションが誤っています。正しいコマンド引数およびオプションを指定して再度実行してください。

KFCA30972-E

ファイル aa....aa **が存在しません。**(E)

aa....aa：ファイル名

(S) 処理を終了します。

(O) 指定したコマンド引数およびオプションが誤っています。ファイル名の指定またはオプションに誤りがないか確認してください。

KFCA30973-E

ファイル aa....aa **にトレースデータがありません。**(E)

aa....aa：ファイル名

(S) 処理を終了します。

(O) ファイル名の指定に誤りがないか確認してください。

KFCA30974-W

トレースデータに不正な種別コードがあります。(E)

(S) 処理を続行します。

(O) OpenTP1 管理者に連絡してください。

KFCA30975-I

使用方法：mqcapiout [{-i **スレッド** ID | -k **コネクションハンドル** | -x | -s | -t }]{ **トレースファイル名** | -c **コアファイル名** }(E)

(S)mqcapiout コマンドの使用方法を示します。

KFCA30976-E

ファイル名が不正です。(E)

(S) 処理を終了します。

(O) ファイル名の指定に誤りがないか確認してください。

KFCA30977-E

aa....aa **のトレースデータが破壊されています。**(E)

aa....aa：ファイル名

(S) 処理を終了します。

(O) OpenTP1 管理者に連絡してください。

KFCA30978-E

the same variable name aa....aa is duplicated in the environment variable definition.（R）

環境変数定義で，同一の変数名 aa....aa が重複して設定されています。

aa....aa：環境変数名

(S)API 実行がエラーリターンします。

〔対策〕環境変数が重複して設定されていないか確認してから再度実行してください。

KFCA30979-E

an I/O error occurred in the file aa....aa. (bb....bb, cc....cc, dd....dd, ee....ee).（R）

ファイルのアクセスで，入出力エラーが発生しました。

aa....aa：入出力エラーが発生したキューファイル名

bb....bb：エラー要因となった関数名

cc....cc：エラー要因となった関数のリターンコード

dd....dd：エラーを出力したモジュール ID

ee....ee：ライン番号

(S) 処理を中断します。

〔対策〕ディスク装置に異常がないか確認してください。

KFCA30980-W

the transaction was changed to rollback-only. reason = aa....aa, XID = [bb....bb][cc....cc]（R）

トランザクションは rollback-only に変更されました。

aa....aa：理由コード

1：サーバとの回線が切断されました。

bb....bb：XID 構造体の data フィールドのうち，先頭から gtrid_length フィールドで指定した長さ分の情報（最大 32 バイト分）

cc....cc：XID 構造体の data フィールドのうち，gtrid_length フィールドで指定したバイトから bqual_length フィールドで指定した長さ分の情報（最大 32 バイト分）

(S) 処理を続行します。

KFCA30981-E

an error occurred by opening the file. file = aa....aa, errno = bb....bb (R)

ファイルのオープンでエラーが発生しました。

aa....aa：エラーが発生したファイル名

bb....bb：エラー番号（open システムコールの errno）

(S)API 実行，コマンド実行がエラーリターンします。

(O) エラー番号に従ってエラー原因を取り除いてから，再度実行してください。

KFCA30982-E

ファイル aa...aa **にコネクションハンドルを持つトレースデータがありません。**(E)

aa....aa：ファイル名

(S) 処理を終了します。

(O)mqcapiout コマンドの -s，-k オプションを使用しないで編集してください。

KFCA31300-E

a system error occurred. location = aa....aa reason = bb....bb（R）

MQ システムでエラーが発生しました。

aa....aa：エラーの発生個所

bb....bb：エラーの発生理由

(S) 例外を発行して処理を中断します。

(O)OpenTP1 管理者に連絡してください。

〔**対策**〕保守員に連絡してください。

KFCA31301-E

the temporary queue is in use. The object that is using the queue = aa....aa（R）

動的キューは使用中です。

aa....aa：キューを使用しているオブジェクトの名前

(S) 例外を発行して処理を中断します。

〔**対策**〕動的キューが使われていないことを確認してから再度実行してください。

KFCA31302-E

an error occurred in MQOPEN. reason code = aa....aa（R）

MQOPEN 命令でエラーが発生しました。

aa....aa：MQOPEN 命令の理由コード

(S) 例外を発行して処理を中断します。

(O)OpenTP1 管理者に連絡してください。

〔**対策**〕cmqc.h の MQI ヘッダファイルと，マニュアル「TP1/Message Queue プログラム作成リファレンス」から理由コード（MQRC*）を調査し，処置してください。

KFCA31303-E

an error occurred in MQCLOSE. reason code = aa....aa（R）

MQCLOSE 命令でエラーが発生しました。

aa....aa：MQCLOSE 命令の理由コード

(S) 例外を発行して処理を中断します。

(O) OpenTP1 管理者に連絡してください。

〔対策〕cmqc.h の MQI ヘッダファイルと，マニュアル「TP1/Message Queue プログラム作成リファレンス」から理由コード（MQRC*）を調査し，処置してください。

KFCA31304-E

an error occurred in MQPUT. reason code = aa....aa（R）

MQPUT 命令でエラーが発生しました。

aa....aa：MQPUT 命令の理由コード

(S) 例外を発行して処理を中断します。

(O) OpenTP1 管理者に連絡してください。

〔対策〕cmqc.h の MQI ヘッダファイルと，マニュアル「TP1/Message Queue プログラム作成リファレンス」から理由コード（MQRC*）を調査し，処置してください。

KFCA31305-E

an error occurred in MQPUT1. reason code = aa....aa（R）

MQPUT1 命令でエラーが発生しました。

aa....aa：MQPUT1 命令の理由コード

(S) 例外を発行して処理を中断します。

(O) OpenTP1 管理者に連絡してください。

〔対策〕cmqc.h の MQI ヘッダファイルと，マニュアル「TP1/Message Queue プログラム作成リファレンス」から理由コード（MQRC*）を調査し，処置してください。

KFCA31306-E

an error occurred in MQGET. reason code = aa....aa（R）

MQGET 命令でエラーが発生しました。

aa....aa：MQGET 命令の理由コード

(S) 例外を発行して処理を中断します。

(O) OpenTP1 管理者に連絡してください。

〔対策〕cmqc.h の MQI ヘッダファイルと，マニュアル「TP1/Message Queue プログラ

ム作成リファレンス」から理由コード（MQRC*）を調査し，処置してください。

KFCA31307-W

message reception is prohibited.（R）

メッセージの受信は禁止されています。

(S) 処理を続行します。

〔対策〕コネクションの受信禁止状態を解除してから受信を実行してください。

KFCA31308-W

an error occurred during the closing of QueueBrowser. contents: aa....aa（R）

QueueBrowser のクローズ処理中にエラーが発生しました。

aa....aa：エラー内容

(S) 処理を続行します。

〔対策〕このメッセージの直前にメッセージ KFCA31303-E が出力された場合，メッセージ KFCA31303-E の指示に従って処置してください。

KFCA31309-W

an error occurred during the hasMoreElements() method processing of an Enumeration object acquired by the getEnumeration method of QueueBrowser.（R）

QueueBrowser の getEnumeration メソッドで取得した Enumeration オブジェクトの hasMoreElements メソッドの処理中にエラーが発生しました。

(S) 処理を続行します。

〔対策〕このメッセージの直前にメッセージ KFCA31306-E が出力された場合，メッセージ KFCA31306-E の指示に従って処置してください。

KFCA31310-W

an error occurred during the nextElement() method processing of an Enumeration object acquired by the getEnumeration method of QueueBrowser.（R）

QueueBrowser の getEnumeration メソッドで取得した Enumeration オブジェクトの nextElement メソッドの処理中にエラーが発生しました。

(S) 処理を続行します。

〔対策〕このメッセージの直前にメッセージ KFCA31306-E が出力された場合，メッセージ KFCA31306-E の指示に従って処置してください。

KFCA31311-W

an error occurred during the deleting of TemporaryQueue. contents: aa....aa（R）

TemporaryQueue の削除処理中にエラーが発生しました。

aa....aa：エラー内容

(S) 処理を続行します。

〔対策〕 このメッセージの直前にメッセージ KFCA31303-E が出力された場合，メッセージ KFCA31303-E の指示に従って処置してください。

KFCA31312-W

a warning occurred in MQPUT. reason code = aa....aa（R）

MQPUT 命令は部分的に完了しました。

aa....aa：MQPUT 命令の理由コード

(S) 処理を続行します。

〔対策〕 cmqc.h の MQI ヘッダファイルと，マニュアル「TP1/Message Queue プログラム作成リファレンス」から理由コード（MQRC*）を調査し，処置してください。

KFCA31313-W

a warning occurred in MQPUT1. reason code = aa....aa（R）

MQPUT1 命令は部分的に完了しました。

aa....aa：MQPUT1 命令の理由コード

(S) 処理を続行します。

〔対策〕 cmqc.h の MQI ヘッダファイルと，マニュアル「TP1/Message Queue プログラム作成リファレンス」から理由コード（MQRC*）を調査し，処置してください。

KFCA31320-I

the environment variable definition was read. aa....aa=bb....bb（R）

環境変数定義を読み込みました。

aa....aa：環境変数名

bb....bb：指定値

(S) 処理を続行します。

KFCA31321-W

an error occurred during environment variable definition reading. aa....aa:bb....bb（R）

環境変数定義の読み込みでエラーが発生しました。

aa....aa：環境変数名

bb....bb：理由コード

-5：パラメタが不正です。環境変数名が DCMQCCLTPORT 以外の場合はデフォル

ト値が設定されます。環境変数名が DCMQCCLTPORT の場合は指定値を使用します。

上記以外：システムエラー

(S) 処理を続行します。ただし指定した値は無効となります。

〔対策〕理由コードを参照し，必要な場合は一度アンデプロイしてパラメタを見直してから再度実行してください。理由コードがシステムエラーの場合は，OpenTP1 管理者に連絡してください。

KFCA31322-E

a logical contradiction error occurred. contents:aa....aa（R）

システム内で論理矛盾によるエラーが発生しました。

aa....aa：エラー内容

(S) 例外を発行して処理を中断します。

〔対策〕保守員に連絡してください。

KFCA31323-I

the connection to the MQC server succeeded. connection handle=aa....aa（R）

MQC サーバとのコネクションを確立しました。

aa....aa：コネクションハンドル

(S) 処理を続行します。

KFCA31324-E

an error occurred during connection to the MQC server. reason=aa....aa（R）

MQC サーバとのコネクションの確立でエラーが発生しました。

aa....aa：MQCONN 命令の理由コード

(S) 例外を発行して処理を中断します。

〔対策〕マニュアル「TP1/Message Queue プログラム作成リファレンス」の MQCONN 命令の説明を参照して障害要因を特定してください。障害の要因を取り除いてから再度アプリケーションを実行してください。

KFCA31325-I

disconnection from the MQC server succeeded. connection handle=aa....aa（R）

MQC サーバとのコネクションを切断しました。

aa....aa：コネクションハンドル

(S) 処理を続行します。

KFCA31326-W

an error occurred during disconnection from the MQC server. handle=aa....aa reason=bb....bb(R)

MQC サーバとのコネクションの切断でエラーが発生しました。

aa....aa：コネクションハンドル

bb....bb：MQDISC 命令の理由コード

(S) 処理を続行します。

〔対策〕 マニュアル「TP1/Message Queue プログラム作成リファレンス」の MQDISC 命令の説明を参照して障害要因を取り除いてください。

MQC サーバの MQC ゲートウェイサーバ監視タイマ値（mqcenv 定義コマンドの -g オプションで指定）が短い場合，このメッセージが出力されることがあります。この場合は MQC ゲートウェイサーバ監視タイマ値を変更して再度実行してください。

KFCA31327-W

a connection to the MQC server was found to be disconnected. connection handle=aa....aa（R）

MQC サーバとのコネクションの切断を検知しました。

aa....aa：コネクションハンドル

(S) 処理を続行します。

〔対策〕 MQC サーバの MQC ゲートウェイサーバ監視タイマ値（mqcenv 定義コマンドの -g オプションで指定）が短い場合，このメッセージが頻繁に出力されることがあります。この場合は MQC ゲートウェイサーバ監視タイマ値を変更して再度実行してください。

KFCA31328-W

an error occurred during QueueManager closing. reason=aa....aa（R）

キューマネジャのクローズ処理でエラーが発生しました。

aa....aa：MQCLOSE 命令の理由コード

(S) 処理を続行します。

〔対策〕 保守員に連絡してください。

KFCA31329-E

an error occurred during local transaction starting. reason=aa..aa（R）

ローカルトランザクションの開始処理でエラーが発生しました。

aa....aa：MQBEGIN 命令の理由コード

(S) 例外を発行して処理を中断します。

〔対策〕 保守員に連絡してください。

KFCA31330-E

an error occurred during local transaction committing. reason=aa..aa（R）

ローカルトランザクションのコミット処理でエラーが発生しました。

aa....aa：MQCMIT 命令の理由コード

(S) 例外を発行して処理を中断します。

〔対策〕 保守員に連絡してください。

KFCA31331-E

an error occurred during local transaction rollback. reason=aa..aa（R）

ローカルトランザクションのロールバック処理でエラーが発生しました。

aa....aa：MQBACK 命令の理由コード

(S) 例外を発行して処理を中断します。

〔対策〕 保守員に連絡してください。

KFCA31332-E

multiple APIs were simultaneously issued within the same QueueSession.（R）

同一の QueueSession 内で複数の API が同時に発行されました。

(S) 例外を発行して処理を中断します。

〔対策〕 同一の QueueSession 内のオブジェクトを使用して，複数のスレッドから API を発行している可能性があります。アプリケーションを見直して再度実行してください。

KFCA31333-W

an error occurred during QueueSession closing. contents:aa..aa（R）

QueueSession のクローズ処理中にエラーが発生しました。

aa....aa：エラー内容

(S) 例外を発行して処理を中断します。
MQC サーバとのコネクションは切断されます。

〔対策〕 保守員に連絡してください。

KFCA31334-W

queue reopening failed during the associate processing of QueueSession. object:aa....aa（R）

QueueSession の associate の処理中にキューの再オープンに失敗しました。

aa....aa：再オープンに失敗したオブジェクトのハッシュコード

(S) 処理を続行します。

〔対策〕 保守員に連絡してください。

KFCA31335-W

queue closing failed during the associate processing of QueueSession. object:aa..aa（R）

QueueSession の associate の処理中にキューのクローズに失敗しました。

aa....aa：クローズに失敗したオブジェクトのハッシュコード

(S) 処理を続行します。

〔対策〕 保守員に連絡してください。

KFCA31336-W

QueueConnection closing failed because QueueSession closing failed. object:aa..aa（R）

QueueConnection のクローズ処理中に QueueSession のクローズに失敗しました。

aa....aa：クローズに失敗したオブジェクトのハッシュコード

(S) 処理を続行します。

〔対策〕 保守員に連絡してください。

KFCA31340-W

(MDB)the queue surveillance processing of MDB has already started.（R）

Message-Driven Bean キュー監視機能は，すでに開始しています。

(S) 処理を続行します。

KFCA31341-W

(MDB) an error occurred during the processing of the queue monitoring thread. MQI instruction name = aa....aa, reason code = bb....bb, maintenance code = cc....cc, thread = dd....dd（R）

キュー監視スレッドでエラーが発生しました。

aa....aa：エラーが発生した MQI 命令名

bb....bb：MQI 命令の理由コード

cc....cc：保守コード

dd....dd：エラーが発生したスレッド名

(S) このメッセージの直後にメッセージ KFCA31345-W が出力された場合，エラーが発生したスレッドを停止します。その他の場合は処理を続行します。

〔対策〕 マニュアル「TP1/Message Queue プログラム作成リファレンス」のエラーが発生した MQI 命令の説明を参照して障害要因を取り除いてください。

KFCA31342-I

(MDB) the processing of the queue monitoring thread will now start. required number of threads = aa....aa（R）

キュー監視スレッドの処理を開始します。

aa....aa：要求スレッド数

(S) 処理を続行します。

KFCA31343-I

(MDB) a thread has started. thread = aa....aa（R）

キュー監視スレッドが開始しました。

aa....aa：開始したスレッド名

(S) 処理を続行します。

KFCA31344-I

(MDB) a thread has stopped. thread = aa....aa（R）

キュー監視スレッドが正常に停止しました。

aa....aa：正常に停止したスレッド名

(S) 処理を続行します。

KFCA31345-W

(MDB) a thread has abnormally stopped. thread = aa....aa（R）

キュー監視スレッドが異常停止しました。

aa....aa：異常停止したスレッド名

(S) 処理を続行します。

〔**対策**〕このメッセージの直前に出力されたメッセージ KFCA31341-W に従って障害要因を取り除いてから，アンデプロイ後に再度デプロイしてください。

KFCA31346-E

(MDB) all threads have stopped.（R）

すべてのキュー監視スレッドが停止しました。

(S)Message-Driven Bean アプリケーションにメッセージが通知されません。

〔**対策**〕異常停止したスレッドの障害要因を取り除いてから，アンデプロイ後に再度デプロイしてください。

付録

付録 A　理由コード

付録 B　JMS 仕様と MQC クライアント機能の JMS インタフェースとの差異

付録 C　MQMessage クラスで変換できるコード化文字セット識別子の一覧

付録 D　用語解説

付録 A　理由コード

TP1/Message Queue - Access では，マニュアル「TP1/Message Queue プログラム作成リファレンス」に記載されている理由コードのほかに，表 A-1 で示す理由コードが出力される可能性があります。理由コードの詳細については，上記のマニュアルを参照してください。

また，次の理由コードでは，マニュアル「TP1/Message Queue プログラム作成リファレンス」に記載されている以外の要因である可能性があります。

- MQRC_Q_MGR_NOT_AVAILABLE
- MQRC_UNEXPECTED_ERROR

理由コード一覧を，次の表に示します。

表 A-1　理由コード一覧

値（10 進数）	値（16 進数）	説明
2058L	X'0000080A'	MQRC_Q_MGR_NAME_ERROR MQCONN 命令で Name 引数に指定された値が不正か，または未知の値です。 要因 次の場合に，この理由コードを返します。 • ポインタ引数が不正な場合 不正なポインタ引数を検出できないことがあります。検出されないときの動作は保証しません。 • WebSphere MQ クライアントアプリケーションが WebSphere MQ クライアントキューマネジャグループ内のキューマネジャに接続した場合で，次に示すどちらかのとき ・キューマネジャグループがサポートされていないとき ・指定された名称を持つキューマネジャグループがないとき 対処 影響がなければ，Name 引数に空白を指定してください。または，使用中の名称が正しいか確認してください。 環境変数に DCMQCDEFCON=Y を設定し，非 XA ライブラリを使用している AP では Name 引数に空白を指定しないでください。 接続先情報定義ファイルに設定しているキューマネジャ名のどちらかを指定してください。

値（10 進数）	値（16 進数）	説明
2059L	X'0000080B'	MQRC_Q_MGR_NOT_AVAILABLE 要因 MQC ゲートウェイサーバに空きがない場合，MQCONN 命令がこの理由コードを返します。 MQC サーバと MQC クライアントのバージョンが対応していない場合，MQCONN 命令がこの理由コードを返す場合があります。 MQDISC 命令発行時に MQC ゲートウェイサーバで後処理をするため，直後に発行された MQCONN 命令がこの理由コードを返す場合があります。 対処 必要な数の MQC ゲートウェイサーバが起動しているかどうか確認してください。 MQC サーバと MQC クライアントが対応バージョンであるかどうか確認してください。 MQDISC 命令 /MQCONN 命令を繰り返し発行する UAP を使用した場合は，MQCONN 命令のリトライ発行処理をしてください。
2195L	X'00000893'	MQRC_UNEXPECTED_ERROR 要因 次のどれかの場合，この理由コードを返すときがあります。 • 環境変数の設定に問題があります。 • MQC サーバの環境に問題があります。 • XA 連携時，xa_open 関数発行前に MQCONN 命令を呼び出しています。 • XA 連携時，TP1/Message Queue - Access 05-00 未満でトランザクションを子ブランチで使用しています。 • XA 連携時，TP1/Message Queue - Access 05-00 以降で環境変数に DCMQCEXPTRN=N を設定して，トランザクションを子ブランチで使用しています。 対処 システムログにエラーメッセージが出力されている場合，出力されたメッセージに従って障害要因を取り除いてください。 MQC サーバのログにエラーメッセージが出力されている場合，出力されたメッセージに従って MQC サーバの障害要因を取り除いてください。 XA 連携をしている場合，xa_open 関数発行前に MQCONN 命令を呼び出しをしていないことを確認してください。 XA 連携時，次の条件の場合はトランザクションを子ブランチでは使用できません。親ブランチで使用してください。 • TP1/Message Queue - Access 05-00 未満 • TP1/Message Queue - Access 05-00 以降で環境変数に DCMQCEXPTRN=N を設定したとき TP1/Message Queue - Access 05-00 以降でトランザクションを子ブランチで使用する場合は環境変数に DCMQCEXPTRN=Y を設定してください。

値（10 進数）	値（16 進数）	説明
6100L	X'000017D4'	MQRC_REOPEN_EXCL_INPUT_ERROR 排他入力用キューのため，再オープンできません。 要因 オープンオブジェクトに正しい ImqObject open options がなく，一つまたは複数の追加オプションが必要です。暗黙の再オープンが必要ですが，キューが排他的入力用にオープンになっているためにクローズできませんでした。また，クローズすると，キューが現在アクセス権のあるプロセスまたはスレッドによって再オープンされる前に，別のプロセスまたはスレッドによってアクセスされる可能性があります。 対処 暗黙の再オープンが必要でなくなるように，発生する可能性のある事態をすべて扱えるようにするため，open options を明示的に設定します。
6101L	X'000017D5'	MQRC_REOPEN_INQUIRE_ERROR 特性の検査が必要なため，再オープンできません。 要因 オープンオブジェクトに正しい ImqObject open options がなく，一つまたは複数の追加オプションが必要です。暗黙の再オープンが必要ですが，オブジェクトの一つまたは複数の特性をクローズ前に動的に検査する必要があるため，クローズできませんでした。また，open options に MQOO_INQUIRE が組み込まれていません。 対処 MQOO_INQUIRE を組み込むよう，open options を明示的に設定します。
6102L	X'000017D6'	MQRC_REOPEN_SAVED_CONTEXT_ERR 状態を保存するため，再オープンできません。 要因 オープンオブジェクトに正しい ImqObject open options がなく，一つまたは複数の追加オプションが必要です。暗黙の再オープンが必要ですが，キューが MQOO_SAVE_ALL_CONTEXT でオープンになっているためにクローズができませんでした。または消去される GET が直前に実行されています。これによって，保存された状態情報がオープンキューと関連づけられるため，この情報はクローズによって破棄されます。 対処 暗黙の再オープンが必要でなくなるように，発生する可能性のある事態をすべて扱えるようにするため，open options を明示的に設定します。
6103L	X'000017D7'	MQRC_REOPEN_TEMPORARY_Q_ERROR 一時キューのため，再オープンできません。 要因 オープンオブジェクトに正しい ImqObject open options がなく，一つまたは複数の追加オプションが必要です。暗黙の再オープンが必要ですが，キューが定義タイプ MQQDT_TEMPORARY_DYNAMIC のローカルキューであるため，クローズができませんでした。このローカルキューはクローズによって破棄されます。 対処 暗黙の再オープンが必要でなくなるように，発生する可能性のある事態をすべて扱えるようにするため，open options を明示的に設定します。

値（10 進数）	値（16 進数）	説明
6104L	X'000017D8'	MQRC_ATTRIBUTE_LOCKED 属性がロックされています。 要因 あるオブジェクトがオープンになっているとき，または ImqQueueManager オブジェクトの場合はそのオブジェクトが接続されているときに，そのオブジェクトの属性の値を変更しようとしました。このような場合，特定の属性を変更できません。属性値を変更する前に，必要に応じてオブジェクトのクローズまたは切断をしてください。 MQINQ コールを実行するために，オブジェクトが予想に反して，しかも暗黙的に接続されたか，オープンされたか，またはその両方が行われた可能性があります。どのメソッド呼び出しによって，MQINQ コールが発生したかどうかを判別してください。 対処 ImqObject open options に MQOO_INQUIRE を組み込み，それらのオプションをもっと早い時期に設定します。
6105L	X'000017D9'	MQRC_CURSOR_NOT_VALID ブラウズカーソルがありません。 要因 オープンキューのためのブラウズカーソルが暗黙の再オープンによって最後に使用されてから，無効になっています。 対処 暗黙の再オープンが必要でなくなるように，発生する可能性のある事態をすべて扱えるようにするため，ImqObject open options を明示的に設定します。
6106L	X'000017DA'	MQRC_ENCODING_ERROR コード化に誤りがあります。 要因 メッセージ項目のコード化は，ペーストするために MQENC_NATIVE でなければなりません。
6107L	X'000017DB'	MQRC_STRUC_ID_ERROR 構造体 ID に誤りがあります。 要因 メッセージ項目の構造体 ID は，データポインタで始まる 4 文字から派生したものですが，欠落しているか，その項目がペーストされるオブジェクトのクラスと矛盾しています。
6108L	X'000017DC'	MQRC_NULL_POINTER ヌルポインタが渡されました。 要因 ヌルでないポインタが必要であるか，暗黙に指定されているのにヌルポインタが与えられました。
6109L	X'000017DD'	MQRC_NO_CONNECTION_REFERENCE 接続がありません。 要因 connection reference はヌルです。ImqQueueManager オブジェクトへの接続が必要です。

値（10 進数）	値（16 進数）	説明
6110L	X'000017DE'	MQRC_NO_BUFFER バッファがありません。 要因 使用可能なバッファがありません。ImqCache オブジェクトの場合は，割り当てることができなく，発生するはずのないオブジェクト状態に内部矛盾があることを示しています。
6111L	X'000017DF'	MQRC_BINARY_DATA_LENGTH_ERROR データ長が不正です。 要因 2 進データの長さが，ターゲット属性の長さと矛盾しています。すべての属性にとってゼロが正しい長さです。 • MQ_ACCOUNTING_TOKEN_LENGTH は，accounting token の正しい長さです。 • MQ_CORREL_ID_LENGTH は，correlation id の正しい長さです。 • MQ_GROUP_ID_LENGTH は，group id の正しい長さです。 • MQ_MSG_ID_LENGTH は，message id の正しい長さです。 • MQ_OBJECT_INSTANCE_ID_LENGTH は，instance id の正しい長さです。 • MQ_TRAN_INSTANCE_ID_LENGTH は，transaction instance id の正しい長さです。
6112L	X'000017E0'	MQRC_BUFFER_NOT_AUTOMATIC バッファはシステムによって管理されていません。 要因 ユーザによって定義・管理されるバッファのサイズは変更できません。ユーザ定義バッファは，置換または回収するしかありません。バッファはシステムによって管理されていないと，サイズを変更できません。
6113L	X'000017E1'	MQRC_INSUFFICIENT_BUFFER バッファが不十分です。 要因 データポインタの後ろに，要求を受け入れるだけの使用可能なバッファスペースが十分にありません。これは，バッファのサイズが変更できないためと考えられます。
6114L	X'000017E2'	MQRC_INSUFFICIENT_DATA データが不十分です。 要因 データポインタの後ろに，要求を受け入れるだけのデータが十分にありません。
6115L	X'000017E3'	MQRC_DATA_TRUNCATED データが切り捨てられました。 要因 あるバッファから別のバッファにコピーするときに，データが切り捨てられました。次のどれかの要因が考えられます。 • ターゲットバッファのサイズを変更できません。 • どれかのバッファをアドレス指定するときに問題があります。 • 代替用のバッファと置き換えられてサイズが小さくなっています。

値（10 進数）	値（16 進数）	説明
6116L	X'000017E4'	MQRC_ZERO_LENGTH ゼロの長さが指定されました。 要因 正の長さが必要であるか，または暗黙に指定されているのにゼロの長さが与えられました。
6117L	X'000017E5'	MQRC_NEGATIVE_LENGTH 負の長さが指定されました。 要因 ゼロまたは正の長さが必要であるか，または暗黙に指定されているのに負の長さが与えられました。
6118L	X'000017E6'	MQRC_NEGATIVE_OFFSET 負のオフセットが指定されました。 要因 ゼロまたは正のオフセットが必要であるか，または暗黙に指定されているのに負のオフセットが与えられました。
6119L	X'000017E7'	MQRC_INCONSISTENT_FORMAT フォーマットが矛盾しています。 要因 メッセージ項目の形式が，その項目がペーストされるオブジェクトのクラスと矛盾しています。
6120L	X'000017E8'	MQRC_INCONSISTENT_OBJECT_STATE オブジェクトが矛盾しています。 要因 オープンになっているオブジェクトと，参照される，接続されていない ImqQueueManager オブジェクトとが矛盾しています。
6121L	X'000017E9'	MQRC_CONTEXT_OBJECT_NOT_VALID オブジェクトがありません。 要因 ImqPutMessageOptions context reference が有効な ImqQueue オブジェクトを参照しません。該当するオブジェクトは，直前に破棄されました。
6122L	X'000017EA'	MQRC_CONTEXT_OPEN_ERROR オブジェクトがオープンできません。 要因 ImqPutMessageOptions context reference は ImqQueue オブジェクトを参照しますが，このオブジェクトをオープンしてコンテキストを確立できませんでした。これは，ImqQueue オブジェクトに不適切な open options があるためと考えられます。参照されたオブジェクトの reason code から，原因を調査してください。
6123L	X'000017EB'	MQRC_STRUC_LENGTH_ERROR データ構造体の長さが不正です。 要因 データ構造体の長さが，その内容と矛盾しています。MQRMH の場合は，長さが固定フィールドとすべてのオフセットデータを含めるのに不十分です。

値（10 進数）	値（16 進数）	説明
6126L	X'000017EE'	MQRC_DISTRIBUTION_LIST_EMPTY 配布リストに組み込まれている ImqQueue オブジェクトがありません。 要因 配布リストに組み込まれている ImqQueue オブジェクトがないため，ImqDistributionList が失敗しました。

付録 B　JMS 仕様と MQC クライアント機能の JMS インタフェースとの差異

Sun Microsystems, Inc. が提供する JMS 1.0 と MQC クライアント機能の JMS インタフェースとの機能差を示します。

インタフェース（パッケージ javax.jms）の機能差を次の表に示します。

表 B-1　インタフェースの機能差

インタフェース名	機能差
MapMessage	TP1/Message Queue では Byte 型配列のデータを扱うため，ほかの型のメッセージは未サポートです。
ObjectMessage	
StreamMessage	
TextMessage	
TemporaryTopic	Pub-Sub 型メッセージングは未サポートです。
Topic	
TopicConnection	
TopicConnectionFactory	
TopicPublisher	
TopicSession	
TopicSubscriber	
XAConnection	XA 関連のインタフェースはオプションのため，未サポートです。
XAConnectionFactory	
XAQueueConnection	
XAQueueConnectionFactory	
XAQueueSession	
XASession	
XATopicConnection	
XATopicConnectionFactory	
XATopicSession	
ExceptionListener	未サポートです。

メソッド（パッケージ javax.jms）の機能差を次の表に示します。

表 B-2　メソッドの機能差

インタフェース名	メソッド名	機能差
Message	void **acknowledge**()	確認機能は提供しません。発行後，即座にリターンします。
	void **clearBody**()	—
	void **clearProperties**()	—
	boolean **getBooleanProperty**(java.lang.String name)	String 型から boolean 型への変換に失敗すると，例外 JMSException がスローされます。
	byte **getByteProperty**(java.lang.String name)	String 型から byte 型への変換に失敗すると，例外 JMSException がスローされます。
	double **getDoubleProperty**(java.lang.String name)	String 型から double 型への変換に失敗すると，例外 JMSException がスローされます。
	float **getFloatProperty**(java.lang.String name)	String 型から float 型への変換に失敗すると，例外 JMSException がスローされます。
	int **getIntProperty**(java.lang.String name)	String 型から int 型への変換に失敗すると，例外 JMSException がスローされます。
	java.lang.String **getJMSCorrelationID**()	内部で，MQBYTE24 型から String 型へ変換されます。
	byte[] **getJMSCorrelationIDAsBytes**()	—
	int **getJMSDeliveryMode**()	—
	Destination **getJMSDestination**()	受信したメッセージの場合は受信したキューが返却されます。
	long **getJMSExpiration**()	—
	java.lang.String **getJMSMessageID**()	—
	int **getJMSPriority**()	—
	boolean **getJMSRedelivered**()	MQMD 構造体の BackoutCount フィールドが 1 以上のときは true を返し，0 のときは false を返します。
	Destination **getJMSReplyTo**()	—

インタフェース名	メソッド名	機能差
	long **getJMSTimestamp**()	キューから取得したメッセージの PutDate, PutTime が日付・時間の形式ではない場合, JMSTimestamp には 0 を設定します。 また，PutDate と PutTime のどちらかが空白の場合, JMSTimestamp にはクライアントの現在時刻が設定されます。
	java.lang.String getJMSType()	setJMSType() で指定した値が返ります。
	long **getLongProperty**(java.lang.String name)	String 型から long 型への変換に失敗すると，例外 JMSException がスローされます。
	java.lang.Object **getObjectProperty**(java.lang.String name)	—
	java.util.Enumeration **getPropertyNames**()	—
	short **getShortProperty**(java.lang.String name)	String 型から short 型への変換に失敗すると，例外 JMSException がスローされます。
	java.lang.String **getStringProperty**(java.lang.String name)	—
	boolean **propertyExists**(java.lang.String name)	—
	void **setBooleanProperty**(java.lang.String name, boolean value)	アプリケーション固有プロパティのキーおよび値は TP1/Message Queue のキューに登録されません。
	void **setByteProperty**(java.lang.String name, byte value)	アプリケーション固有プロパティのキーおよび値は TP1/Message Queue のキューに登録されません。
	void **setDoubleProperty**(java.lang.String name, double value)	アプリケーション固有プロパティのキーおよび値は TP1/Message Queue のキューに登録されません。
	void **setFloatProperty**(java.lang.String name, float value)	アプリケーション固有プロパティのキーおよび値は TP1/Message Queue のキューに登録されません。

インタフェース名	メソッド名	機能差
	void **setIntProperty**(java.lang.String name, int value)	アプリケーション固有プロパティのキーおよび値は TP1/Message Queue のキューに登録されません。
	void **setJMSCorrelationID**(java.lang.String correlationID)	内部で，String 型から MQBYTE24 型へ変換されますが，このとき 24 バイトを超える部分は切り捨てられます。
	void **setJMSCorrelationIDAsBytes**(byte[] correlationID)	24 バイトを超える部分は切り捨てられます。
	void **setJMSDeliveryMode**(int deliveryMode)	—
	void **setJMSDestination**(Destination destination)	—
	void **setJMSExpiration**(long expiration)	—
	void **setJMSMessageID**(java.lang.String id)	—
	void **setJMSPriority**(int priority)	—
	void **setJMSRedelivered**(boolean redelivered)	—
	void **setJMSReplyTo**(Destination replyTo)	—
	void **setJMSTimestamp**(long timestamp)	—
	void **setJMSType**(java.lang.String type)	値は TP1/Message Queue のキューに登録されません。
	void **setLongProperty**(java.lang.String name, long value)	アプリケーション固有プロパティのキーおよび値は TP1/Message Queue のキューに登録されません。
	void **setObjectProperty**(java.lang.String name, java.lang.Object value)	アプリケーション固有プロパティのキーおよび値は TP1/Message Queue のキューに登録されません。
	void **setShortProperty**(java.lang.String name, short value)	アプリケーション固有プロパティのキーおよび値は TP1/Message Queue のキューに登録されません。
	void **setStringProperty**(java.lang.String name, java.lang.String value)	アプリケーション固有プロパティのキーおよび値は TP1/Message Queue のキューに登録されません。
BytesMessage（extends **Message**）	int **readBytes**(bute[] value, int length)	length の値が不正な場合，例外 JMSException がスローされます。
MessageProducer	void **close**()	—

インタフェース名	メソッド名	機能差
	int **getDeliveryMode**()	―
	boolean **getDisableMessageID**()	このメソッドで取得した値に関係なく，TP1/Message Queue では常にメッセージ識別子は使用可能です。
	boolean **getDisableMessageTimestamp**()	このメソッドで取得した値に関係なく，TP1/Message Queue では常に登録日時は使用可能です。
	int **getPriority**()	―
	long **getTimeToLive**()	―
	void **setDeliveryMode**(int deliveryMode)	―
	void **setDisableMessageID**(boolean value)	このメソッドで設定した値に関係なく，TP1/Message Queue では常にメッセージ識別子は使用可能です。
	void **setDisableMessageTimestamp**(boolean value)	このメソッドで設定した値に関係なく，TP1/Message Queue では常に登録日時は使用可能です。
	void **setPriority**(int defaultPriority)	―
	void **setTimeToLive**(long timeToLive)	―
QueueSender（extends **MessageProducer**）	全メソッドで機能差はありません。	
MessageConsumer	void **close**()	―
	MessageListener **getMessageListener**()	―
	java.lang.String **getMessageSelector**()	―
	Message **receive**()	―
	Message **receive**(long timeout)	―
	Message **receiveNoWait**()	―
	void **setMessageListener**(MessageListener listener)	メッセージリスナは使用されません。
QueueReceiver（extends **MessageConsumer**）	全メソッドで機能差はありません。	
QueueBrowser	void **close**()	―
	java.util.Enumeration **getEnumeration**()	―

インタフェース名	メソッド名	機能差
	java.lang.String **getMessageSelector**()	—
	public Queue **getQueue**()	—
Queue（extends **Destination**）	全メソッドで機能差はありません。	
TemporaryQueue（extends **Queue**）	全メソッドで機能差はありません。	
ConnectionMetaData	全メソッドで機能差はありません。	
Connection・**QueueConnection**	void **close**()	—
	java.lang.String **getClientID**()	未サポートです。
	ExceptionListener **getExceptionListener**()	未サポートです。
	ConnectionMetaData **getMetaData**()	—
	void **setClientID**(java.lang.String clientID)	未サポートです。
	void **setExceptionListener**(ExceptionListener listener)	未サポートです。
	void **start**()	—
	void **stop**()	—
	ConnectionConsumer **createConnectionConsumer**(Queue queue, java.lang.String messageSelector, ServerSessionPool sessionPool, int maxMessages)	未サポートです。
	QueueSession **createQueueSession**(boolean transacted, int acknowledgeMode)	—
ConnectionFactory・**QueueConnectionFactory**	public QueueConnection **createQueueConnection**()	—
	public QueueConnection **createQueueConnection**(java.lang.String userName,java.lang.String password)	パラメタ userName, password は使用されません。
Session・**QueueSession**	void **close**()	—
	void **commit**()	—
	BytesMessage **createBytesMessage**()	—
	MapMessage **createMapMessage**()	未サポートです。
	Message **createMessage**()	—
	ObjectMessage **createObjectMessage**()	未サポートです。

インタフェース名	メソッド名	機能差
	ObjectMessage **createObjectMessage**(java.io.Serializable object)	未サポートです。
	StreamMessage **createStreamMessage**()	未サポートです。
	TextMessage **createTextMessage**()	未サポートです。
	TextMessage **createTextMessage**(java.lang.String text)	未サポートです。
	MessageListener **getMessageListener**()	－
	boolean **getTransacted**()	－
	void **recover**()	未サポートです。
	void **rollback**()	－
	void **run**()	未サポートです。
	void **setMessageListener**(MessageListener listener)	パラメタ listener の設定はできますが，使用されません。
	QueueBrowser **createBrowser**(Queue queue)	－
	QueueBrowser **createBrowser**(Queue queue, java.lang.String messageSelector)	パラメタ messageSelector は使用できますが，制限があります。詳細については，7 章の「メッセージセレクタ」を参照してください。
	Queue **createQueue**(java.lang.String queueName)	未サポートです。
	QueueReceiver **createReceiver**(Queue queue)	－
	QueueReceiver **createReceiver**(Queue queue, java.lang.String messageSelector)	パラメタ messageSelector は使用できますが，制限があります。詳細については，7 章の「メッセージセレクタ」を参照してください。
	QueueSender **createSender**(Queue queue)	－
	TemporaryQueue **createTemporaryQueue**()	－

（凡例）

－：JMS 仕様との差異はありません。

付録 C　MQMessage クラスで変換できるコード化文字セット識別子の一覧

MQMessage クラスの readString メソッド，writeString メソッドでの文字コード変換ができます。MQMessage クラスで変換できるコード化文字セット識別子（characterSet 変数）の一覧（Java コードセット一覧）を次の表に示します。説明の欄に国名を記載している場合，その国の言語を示しています。

表 C-1　MQMessage クラスで変換できるコード化文字セット識別子の一覧

characterSet 変数の指定	Java コードセット	説明
37	Cp037	米国，カナダ（二カ国語，フランス語），オランダ，ポルトガル，ブラジル，オーストラリア
277	Cp277	IBM デンマーク，ノルウェー
437	Cp437	MS-DOS 米国，オーストラリア，ニュージーランド，南アフリカ
500	Cp500	EBCDIC 500V1
813	ISO8859_7	ISO 8859-7，ラテン／ギリシャ文字アルファベット
819	ISO8859_1	ISO 8859-1，ラテンアルファベット No.1
850	Cp850	MS-DOS ラテン文字 -1
852	Cp852	MS-DOS ラテン文字 -2
866	Cp866	MS-DOS ロシア語
870	Cp870	IBM 多言語ラテン文字 -2
912	ISO8859_2	ISO 8859-2，ラテンアルファベット No.2
915	ISO8859_5	ISO 8859-5，ラテン／キリル文字アルファベット
916	ISO8859_8	ISO 8859-8，ラテン／ヘブライ語アルファベット
920	ISO8859_9	ISO 8859-9，ラテンアルファベット No.5
930	Cp930	UDC 4370 文字を含む日本語カタカナ漢字，5026 のスーパーセット
932	MS932	Windows 日本語
939	Cp939	UDC 4370 文字を含む日本語ラテン文字漢字，5035 のスーパーセット
950	Big5	Big5，中国語（繁体字）
954	EUC_JP	JIS X 0201，0208，0212，EUC エンコーディング，日本語
964	EUC_TW	CNS11643（Plane 1-3），EUC エンコーディング，中国語（繁体字）
970	EUC_KR	KS C 5601，EUC エンコーディング，韓国語

characterSet 変数の指定	Java コードセット	説明
1089	ISO8859_6	ISO 8859-6，ラテン／アラビア文字アルファベット
1200	Unicode	Unicode
1208	UTF8	8 ビット Unicode Transformation Format
1381	Cp1381	IBM OS/2，DOS 中国（中華人民共和国）
33722	Cp33722	IBM-eucJP - 日本語（5050 のスーパーセット）

注 1

表 C-1 以外の characterSet 変数を指定し，readString メソッド，writeString メソッドを発行した場合は，指定した characterSet の先頭に "Cp" を付加したコードセットへ文字コード変換します。

（例）

characterSet 変数に "273" を指定した場合，readString メソッド，writeString メソッド内で，"Cp273" へ文字コード変換します。

"Cp" を付加したコードセットが Java でサポートされていない場合，文字コード変換は失敗します。

コードセットの詳細については Java のエンコーディング仕様を参照してください。

注 2

characterSet 変数に MQCCI_Q_MGR を指定した場合，readString メソッド，または writeString メソッドでは，characterSet 変数に "819" が指定されているものとして文字コード変換します。

付録 D　用語解説

（記号）

$DCCONFPATH

OpenTP1 定義ファイルを格納するディレクトリを完全パス名で指定してある環境変数名です。

$DCDIR

OpenTP1 ディレクトリを完全パス名で指定してある環境変数名です。

（英字）

JMS（Java Message Service）

Java でのメッセージキューイング機能を提供する API です。
TP1/Message Queue - Access の JMS インタフェースは，Sun Microsystems, Inc. が提供する JMS 1.0 のインタフェースに基づいて実装しています。この JMS インタフェースを使用して，TP1/Message Queue と J2EE 準拠のアプリケーションサーバを連携します。なお，TP1/Message Queue - Access が連携できる J2EE 準拠のアプリケーションサーバは Cosminexus だけです。

JTA（Java Transaction API）

J2EE アーキテクチャにトランザクション処理サービスを提供する API です。JTA には，アプリケーションサーバ（Cosminexus）と TP1/Message Queue との通信で必要となる API が定義されています。

RM（Resource Manager）

リソースマネジャのことです。リソースに対するアクセスを提供します。

TM（Transaction Manager）

トランザクションマネジャのことです。トランザクションに対して識別子を割り当て，進行をモニタし，トランザクションの完了と，失敗したときのリカバリをします。

XA インタフェース

TM と RM との間で相互に宣言されるインタフェースです。TM に対し，RM の作業をグローバルトランザクションの中に構成し，決着とリカバリを調停します。

（サ行）

サービス名ファイル

サービス名称とポート番号を対応づけたファイルです。標準的なパス名を次に示します。

- Windows の場合

```
%SystemRoot%¥system32¥drivers¥etc¥services
```

- UNIX の場合

  /etc/services

セグメント

MQC プロトコルでは，UAP で指定されたパラメタを一定の長さに分割してヘッダを付けたあと，TCP/IP に送信します。このときの転送の単位です。

（ハ行）

ホスト名ファイル

ホスト名称と IP アドレスを対応づけたファイルです。標準的なパス名を次に示します。

- Windows の場合

  %SystemRoot%¥system32¥drivers¥etc¥hosts

- UNIX の場合

  /etc/hosts

索引

記号

$DCCONFPATH 414
$DCDIR 414

数字

2 進バイトアレイをカプセル化 125

A

API トレース出力単位 31
API トレースディスク出力要否 31
API トレースファイル 47
API トレースファイル取得 65
API トレースファイル数 31
API トレース容量 31
application.xml 15,17,19,21

B

BytesMessage インタフェース 298

C

C++ インタフェース 121
C++ クラス一覧 123
C++ クラス継承図 124
C++ のサンプルアプリケーション 211
CMQ*.cbl 14,18,20
cmqc.h 14,16,18,20
cmqcfc.h 14,16,18,20
COBOL コピーファイル 20
compileBean 15,17,19
compileBean.bat 21
compileClient 15,17,19
compileClient.bat 21
config.xml 15,17,19,21
ConnectionMetaData インタフェース 307

D

DeliveryMode インタフェース 310
deployApp 15,17,19,21
Destination インタフェース 311

E

ejb-jar.xml 15,17,19,21
Enumeration インタフェース 355

I

imq*.hpp 14,16,18,20
ImqBinary クラス 125
ImqCache クラス 128
ImqDeadLetterHeader クラス 132
ImqError クラス 138
ImqGetMessageOptions クラス 140
ImqHeader クラス 143
ImqItem クラス 145
ImqMessageTracker クラス 155
ImqMessage クラス 147
ImqObject クラス 159
ImqProcess クラス 166
ImqPutMessageOptions クラス 168
ImqQueueManager クラス 188
ImqQueue クラス 171
ImqString クラス 200
ImqTrigger クラス 207
imqtype.h 14,16,18,20

J

JavaEnvironment トレース 47
JavaEnvironment トレース情報の出力レベル 223
JavaEnvironment トレースファイルの出力形式 48
Java インタフェース 215
Java 環境の設定 223
Java クラス一覧 218
Java クラス継承図 219
Java クラスライブラリ 20
Java コンソール 47

Java のサンプルコーディング 277
Java パッケージ 217
JMSAPI トレース情報の出力レベル 33,53
JMSAPI トレースファイル 47
JMSAPI トレースファイルの出力形式 52
JMSPRF トレース情報の取得レベル 33
JMSPRF トレースファイル 47
JMSPRF トレースファイルの取得形式 55
JMSSample1.Client.java 15,17,19
JMSSample1.java 15,17,19,21
JMSSample1Client.java 21
JMSSample1EJB.java 15,17,19,21
JMSSample1Home.java 15,17,19,21
JMSSample2.Client.java 15,17,19
JMSSample2.java 15,17,19,21
JMSSample2Client.java 21
JMSSample2EJB.java 15,17,19,21
JMSSample2Home.java 15,17,19,21
JMS インタフェース 279
JMS インタフェース一覧 282
JMS インタフェース機能 10
JMS インタフェース継承図 284
JMS インタフェースの Java パッケージ 281
JMS インタフェースのサンプルアプリケーション 361
JMS インタフェースのサンプルコーディング 366
JMS インタフェースのメソッドと MQI の対応 286
JMS プロパティと対応する MQMD 構造体のフィールド 288
JMS プロパティ読み込み時の型変換 290
JMS ヘッダと対応する MQMD 構造体のフィールド 288
JMS メッセージのヘッダとプロパティ 287
JP.co.Hitachi.soft.MQ.Access 217
JTA インタフェース接続 7

K

KFCA30950-E 380
KFCA30951-E 380
KFCA30952-E 380
KFCA30953-W 380
KFCA30954-W 381
KFCA30960-E 381
KFCA30961-E 382
KFCA30962-E 382
KFCA30963-E 382
KFCA30964-E 383
KFCA30965-E 383
KFCA30966-E 383
KFCA30967-E 384
KFCA30968-W 384
KFCA30969-E 384
KFCA30970-E 384
KFCA30971-E 384
KFCA30972-E 385
KFCA30973-E 385
KFCA30974-W 385
KFCA30975-I 385
KFCA30976-E 385
KFCA30977-E 385
KFCA30978-E 386
KFCA30979-E 386
KFCA30980-W 386
KFCA30981-E 386
KFCA30982-E 387
KFCA31300-E 387
KFCA31301-E 387
KFCA31302-E 387
KFCA31303-E 388
KFCA31304-E 388
KFCA31305-E 388
KFCA31306-E 388
KFCA31307-W 389
KFCA31308-W 389
KFCA31309-W 389
KFCA31310-W 389
KFCA31311-W 389
KFCA31312-W 390
KFCA31313-W 390
KFCA31320-I 390
KFCA31321-W 390
KFCA31322-E 391
KFCA31323-I 391

KFCA31324-E 391
KFCA31325-I 391
KFCA31326-W 392
KFCA31327-W 392
KFCA31328-W 392
KFCA31329-E 392
KFCA31330-E 393
KFCA31331-E 393
KFCA31332-E 393
KFCA31333-W 393
KFCA31334-W 393
KFCA31335-W 394
KFCA31336-W 394
KFCA31340-W 394
KFCA31341-W 394
KFCA31342-I 394
KFCA31343-I 395
KFCA31344-I 395
KFCA31345-W 395
KFCA31346-E 395

L

libmqc.a 14,16,18
libmqc.dll 20
libmqc.lib 20
libmqc.so 16,18
libmqccb.a 14,18
libmqccb.dll 20
libmqccb.lib 20
libmqccb.so 18
libmqccbx.a 14,18
libmqccbx.dll 20
libmqccbx.lib 20
libmqccbx.so 18
libmqccpp.a 14,16,18
libmqccpp.so 16,18
libmqcj.a 15
libmqcj.so 16,18
libmqcpp.dll 20
libmqcpp.lib 20
libmqcppx.dll 20
libmqcppx.lib 20
libmqcx.a 14,18
libmqcx.dll 20
libmqcx.lib 20
libmqcx.so 18
LIBPATH 14

M

makefile 20
Message-Driven Bean キュー監視機能 10
MessageConsumer インタフェース 327
MessageProducer インタフェース 330
Message インタフェース 312
MQAccessSample.java 15,16,18,20
MQBACK 命令 100
MQBEGIN 命令 103
MQBO 構造体 109
mqc.jar 15,16,18,20
mqcadpt.rar 15,17,19,21
mqcadptdef.jar 15,17,19,21
mqcapiout 15,17,19,65
mqcapiout.exe 21
mqcj.dll 20
MQCMIT 命令 106
mqcsample.c 14,16,18,20
MQCSAMPLE.cbl 14,18,20
mqcsample.cpp 14,16,18,20
MQC インタフェース 264
MQC インタフェース（JMS） 356
MQC クライアント機能 5
MQC クライアント機能の運用コマンド一覧 64
MQC クライアント機能の環境変数のオペランド 28
MQC クライアント機能のセットアップ 14
MQC ゲートウェイサーバ 5
MQC コネクション 3
MQC サーバ機能 5
MQC プロトコル 3
MQC リスナサーバ 5
MQC リスナサーバまたは MQC ゲートウェイサーバへの最大接続試行時間 30
MQDistributionListItem クラス 222
MQDistributionList クラス 220

MQDLH 構造体，および MQRMH 構造体の共通機能をカプセル化 143
MQDLH 構造体の特定の機能をカプセル化 132
MQEnvironment クラス 223
MQException クラス 225
MQGetMessageOptions クラス 233
MQGET 命令の待ち合わせ最大時間 29
MQGMO 構造体をカプセル化 140
MQI 一覧 98
MQManagedObject クラス 235
MQMD 構造体をカプセル化 147
MQMessageTracker クラス 248
MQMessage クラス 237
MQMessage クラスで変換できるコード化文字セット識別子の一覧 412
MQPMO 構造体をカプセル化 168
MQProcess クラス 249
MQPutMessageOptions クラス 251
MQQueue:get メソッドのオプション 233
MQQueue:put メソッドのオプション 251
MQQueueManager クラス 260
MQQueue クラス 253
MQRC_ATTRIBUTE_LOCKED 401
MQRC_BINARY_DATA_LENGTH_ERROR 402
MQRC_BUFFER_NOT_AUTOMATIC 402
MQRC_CONTEXT_OBJECT_NOT_VALID 403
MQRC_CONTEXT_OPEN_ERROR 403
MQRC_CURSOR_NOT_VALID 401
MQRC_DATA_TRUNCATED 402
MQRC_DISTRIBUTION_LIST_EMPTY 404
MQRC_ENCODING_ERROR 401
MQRC_INCONSISTENT_FORMAT 403
MQRC_INCONSISTENT_OBJECT_STATE 403
MQRC_INSUFFICIENT_BUFFER 402
MQRC_INSUFFICIENT_DATA 402
MQRC_NEGATIVE_LENGTH 403
MQRC_NEGATIVE_OFFSET 403
MQRC_NO_BUFFER 402
MQRC_NO_CONNECTION_REFERENCE 401
MQRC_NULL_POINTER 401
MQRC_Q_MGR_NAME_ERROR 398
MQRC_Q_MGR_NOT_AVAILABLE 399
MQRC_REOPEN_EXCL_INPUT_ERROR 400
MQRC_REOPEN_INQUIRE_ERROR 400
MQRC_REOPEN_SAVED_CONTEXT_ERR 400
MQRC_REOPEN_TEMPORARY_Q_ERROR 400
MQRC_STRUC_ID_ERROR 401
MQRC_STRUC_LENGTH_ERROR 403
MQRC_UNEXPECTED_ERROR 399
MQRC_ZERO_LENGTH 403
MQTM 構造体をカプセル化 207
MQ のキューにアクセス 253
MQ のキューマネジャにアクセス 260
MQ のプロセス定義にアクセス 249
MQ の例外発生 225

Q

QueueBrowser インタフェース 338
QueueConfig 15,17,19,21
QueueConnectionFactory インタフェース 344
QueueConnection インタフェース 340
QueueReceiver インタフェース 327
QueueSender インタフェース 330
QueueSession インタフェース 346
Queue インタフェース 311

R

readme.txt 21
RM 関連オブジェクトファイル名 36
RM スイッチ名 36
RM 名 36

S

SHLIB_PATH 14

T

TCP/IP ウィンドウサイズ 31
TemporaryQueue インタフェース 354
TemporaryQueue オブジェクト生成時のモデルキュー定義 33
testClient 15,17,19
testClient.bat 21
TP1/Message Queue - Access の概要 2
TP1/Message Queue - Access の機能 5
TP1/Server Baseと連携する場合の注意事項 25

U

unDeployApp 15,17,19,21

W

Windows を使用する場合の注意事項 11

X

XA インタフェース接続 6
XA インタフェースの設定項目 36
XA インタフェースライブラリ 20

あ

アプリケーション作成時の注意事項（JMS） 294
アプリケーション使用時の注意事項（JMS） 296

い

インストール時に作成されるファイル（AIX） 14
インストール時に作成されるファイル（HP-UX 11i V2.0（IPF64）） 16
インストール時に作成されるファイル（Linux） 18
インストール時に作成されるファイル（Windows） 20

う

運用コマンド 63

え

エラーに関する情報 138

か

環境変数 22
環境変数のオペランド（JMS インタフェース用） 33
環境変数のオペランド（共通） 28
環境変数のオペランド（接続先情報定義ファイルのキューマネジャ接続用） 34
環境変数の設定項目（JMS インタフェース使用時） 24
環境変数の設定場所の有効性 26
監視タイマ 8

き

記述形式 378
キュー定義ファイルの作成（JMS インタフェース用） 34
キュー定義ファイル名 33
キューマネジャおよびキューの属性をカプセル化 159
キューマネジャ構成定義 39
キューマネジャをカプセル化 188

く

クライアントアプリケーション作成時の注意（MQI） 112
クライアントアプリケーションの作成 43
クラス継承図（C++ の場合） 124
クラス継承図（Java の場合） 219
クラスの一覧（C++） 123
クラスの一覧（Java） 218
グローバルトランザクション 105

け

継続セグメント受信監視タイマ値 30

結果応答受信監視タイマ値 29

こ

コアファイル（.core） 47
コネクションプーリング 10
コマンド実行時の注意事項 64

さ

サービス名ファイル 414
サンプル AP 20
サンプルコーディング（COBOL 言語） 116
サンプルコーディング（C 言語） 113

し

時間監視機能 8
自システムの IP アドレス 29
自システムのポート番号 29
システムログファイル 47
出力形式 378
障害時に取得する情報 47
障害対策 47
使用できる MQI 98
処理の流れ（Java の場合） 276
処理の流れ（JMS インタフェースの場合） 362

せ

セグメントサイズ 30
接続先情報定義ファイルの設定項目 38

て

デプロイおよびアンデプロイ時の注意事項（JMS インタフェース使用時） 24

と

トランザクション処理方式の拡張要否 32
トランザクション連携 6
トリガできるアプリケーションプロセスをカプセル化 166

ひ

非 XA インタフェース接続 6
非 XA インタフェースパッケージ 217
非 XA インタフェースライブラリ 20

へ

ヘッダファイル 20

ほ

ホスト名ファイル 415

ま

窓口となる MQC リスナサーバの IP アドレス 28
窓口となる MQC リスナサーバのサービス名 28
窓口となる MQC リスナサーバのポート番号 29
窓口となる MQC リスナサーバのホスト名 28

め

メッセージ ID の記号の説明 378
メッセージ一覧 380
メッセージキューをカプセル化 171
メッセージセレクタ 292
メッセージデータを構築 147
メッセージ内の一つの項目 145
メッセージの形式 378

り

リンケージオプション（C++ 言語） 44
リンケージオプション（COBOL 言語） 43
リンケージオプション（C 言語） 43

れ

例外クラスの一覧（JMS インタフェース） 283

ろ

ローカルトランザクション 104
ローカルトランザクション開始オプション 109
ローカルトランザクションの開始 103
ローカルトランザクションのコミット 106
ローカルトランザクションのロールバック 100
ローカルトランザクションを使用する場合の処理の流れ 111

ソフトウェアマニュアルのサービス ご案内

ソフトウェアマニュアルについて，３種類のサービスをご案内します。ご活用ください。

１．マニュアル情報ホームページ

ソフトウェアマニュアルの情報をインターネットで公開しております。

URL　http://www.hitachi.co.jp/soft/manual/

ホームページのメニューは次のとおりです。

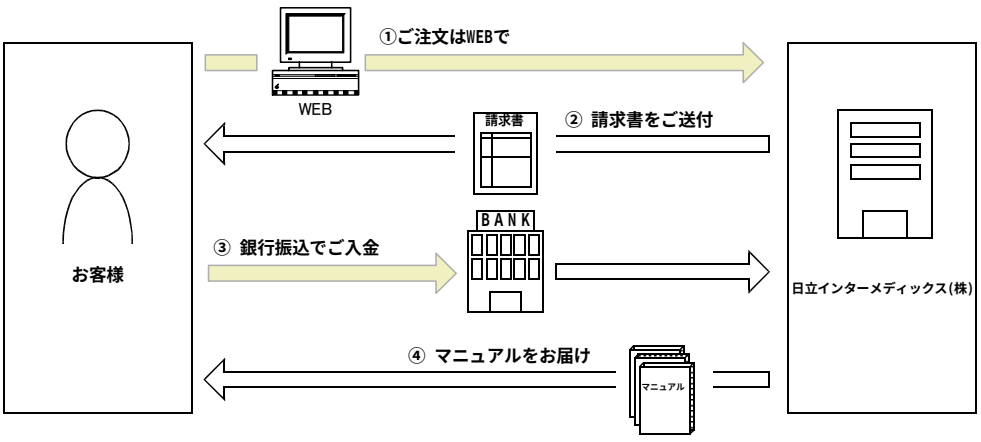
■マニュアル一覧	日立コンピュータ製品マニュアルを製品カテゴリ，マニュアル名称，資料番号のいずれかから検索できます。
■CD-ROMマニュアル情報	複数マニュアルを格納したCD-ROMマニュアルを提供しています。どの製品に対応したCD-ROMマニュアルがあるか，を参照できます。
■マニュアルのご購入	日立インターメディックス(株)の「日立コンピュータ製品マニュアルサイト」からお申し込みできます。 (詳細は「３．マニュアルのご注文」を参照してください。)
■Web提供マニュアル一覧	インターネットで参照できるマニュアルの一覧を提供しています。 (詳細は「２．インターネットからのマニュアル参照」を参照してください。)
■ご意見・お問い合わせ	マニュアルに関するご意見，ご要望をお寄せください。

２．インターネットからのマニュアル参照(ソフトウェアサポートサービス)

ソフトウェアサポートサービスの契約をしていただくと，インターネットでマニュアルを参照できます。
本サービスの対象となる契約の種別，及び参照できるマニュアルは，マニュアル情報ホームページでご確認ください。なお，ソフトウェアサポートサービスは，マニュアル参照だけでなく，対象製品に対するご質問への回答，問題解決支援，バージョン更新版の提供など，お客様のシステムの安定的な稼働のためのサービスをご提供しています。まだご契約いただいていない場合は，ぜひご契約いただくことをお勧めします。

３．マニュアルのご注文

日立インターメディックス(株)の「日立コンピュータ製品マニュアルサイト」からご注文ください。



① 下記 URL にアクセスして必要事項を入力してください。

URL　http://www2.himdx.net/manual/privacy.asp?purchase_flag=1

② ご注文いただいたマニュアルについて，請求書をお送りします。

③ 請求書の金額を指定銀行へ振り込んでください。なお，送料は弊社で負担します。

④ 入金確認後，７日以内にお届けします。在庫切れの場合は，納期を別途ご案内いたします。