MQSeries® Everyplace for Multiplatforms

プログラミング・リファレンス

バージョン 1.2

SC88-8655-02
(英文原典：SC34-5846-02)

MQSeries® Everyplace for Multiplatforms

プログラミング・リファレンス

バージョン 1.2

SC88-8655-02
(英文原典：SC34-5846-02)

ご注意

本書の情報およびそれによってサポートされる製品を使用する前に、 505ページの『付録. 特記事項』に記載する一般情報をお読みください。

ライセンスについての警告

MQSeries Everyplace for Multiplatforms バージョン 1.2 ツールキットにより、開発者は MQSeries Everyplace アプリケーションを作成し、それを実行するための環境を作成することができます。

この製品またはこの製品を使用するアプリケーションを実稼働環境において展開する前に、必要なライセンスを必ず取得してください。

MQSeries Everyplace for Multiplatforms バージョン 1.2 ツールキットにより、開発者は MQSeries Everyplace アプリケーションを作成し、それを実行するための環境を作成することができます。

この製品およびこの製品を使用するアプリケーションを実稼働環境において展開する前に、必要なライセンスを必ず取得してください。

指定したサーバー・プラットフォームで MQSeries Everyplace を使用するには (コードの開発およびテストの目的以外で)、 capacity-unit 使用許可 (ライセンス証書に記載され、発行された capacity-unit と pricing group table に従った MQSeries Everyplace の使用に対して、サポートが有効になります) を、マシンおよびマシンのアップグレードごとのプログラム使用ライセンスの交付を受けるために、取得しなければなりません。

device platform 使用許可 (ライセンス証書に記載され、MQSeries Everyplace の使用に対してサポートが有効になります) は、指定したクライアント・プラットフォームで、この製品を使用する (コードの開発およびテストの目的以外で) ために必要です。これらのライセンスにより、ユーザーが MQSeries Everyplace Bridge を使用すること、または IBM が発行し、Web の下記 URL においても使用可能な MQSeries Everyplace の pricing group リストで指定されたサーバー・プラットフォームで実行することが認められるものではありません。

これらの制約事項の詳細については、http://www.ibm.com/software/mqseries を参照してください。

本書は、随時改訂され、内容は更新されます。最新の版については、MQSeries ファミリー・ライブラリーの Web ページ `http://www.ibm.com/software/ts/mqseries/library/` をご覧ください。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

`http://www.ibm.com/jp/manuals/main/mail.html`

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

`http://www.ibm.com/jp/manuals/` の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

原　典： SC34-5846-02
MQSeries® Everyplace for Multiplatforms
Programming Reference
Version 1.2

発　行： 日本アイ・ビー・エム株式会社

担　当：　ナショナル・ランゲージ・サポート

第1刷 2001.5

目次

本書について

本書は、MQSeries Everyplace for Multiplatforms 製品 (本書では一般に MQSeries Everyplace と呼びます) のプログラミング・リファレンスであり、 MQSeries Everyplace クラス・ライブラリーに含まれているさまざまなメソッドのパラメーターおよび呼び出しシーケンスについて詳しく解説しています。本書は、MQSeries Everyplace for Multiplatforms プログラミング・ガイドに加え、 MQSeries Everyplace プログラムを作成するのに使用するプログラム言語に関する資料またはマニュアルとともにお使いください。

本書は、随時改訂され、内容は更新されます。最新の版については、MQSeries ファミリー・ライブラリーの Web ページ `http://www.ibm.com/software/ts/mqseries/library/` をご覧ください。

本書の対象読者

本書は、パーベイシブ・コンピューティング環境で使用される MQSeries Everyplace プログラムを作成するプログラマーを対象としています。

前提条件となる知識

本書は、作成する MQSeries Everyplace プログラムの言語に関する基本的なプログラミング手法の実用的な知識を、読者がすでに持っていることを前提としています。

安全なメッセージングの基本的な概念を理解していると役立ちます。この点についてご理解いただく上で、次の MQSeries 資料が参考になります。

- *MQSeries An Introduction to Messaging and Queuing*
- *MQSeries (Windows NT® 版) インストールの手引き V5.1*

これらの資料は、オンラインの MQSeries ライブラリーの『Book』セクションから、ソフトコピーの形で利用できます。 MQSeries の Web サイト (URL アドレス `http://www.ibm.com/software/ts/MQSeries/library/`) から、この資料を利用することもできます。

変更の要約

このセクションでは、この版の「*MQSeries Everyplace for Multiplatforms プログラミング・リファレンス*」に加えられた変更点について説明します。前の版から変更された箇所については、変更箇所の左側に縦線が記されています。

この版 (SC88-8655-02) に加えられた変更

情報の修正や明瞭化に加えて、以下に関する情報が追加されました。

- MQeAbstractMessageStore
- MQeReducedDiskFieldsAdapter

直前の版 (SC88-8655-01) に加えられた変更

情報の修正や明瞭化に加えて、以下に関する情報が追加されました。

- さまざまなアダプター・クラス
- QueueRule クラスに追加された filterMessage メソッド
- 例外のリスト

第1部 MQSeries Everyplace Java API

第1章 MQSeries Everyplace クラスおよびインターフェース

これ以降の章では、 MQSeries Everyplace とともに提供されるクラスおよびインターフェースについて詳しく解説されます。取り上げられるクラスの順番は、出荷されるパッケージでのアルファベット順になっています。

MQSeries Everyplace に含まれているパッケージを以下に示します。

com.ibm.mqe

表 1. com.ibm.mqe パッケージのクラス

クラス名	目的
MQe	他の MQSeries Everyplace クラスを派生させます。
MQeAbstractMessageStore	メッセージ・ストレージ・インターフェースを定義します。
MQeAdapter	これは、すべての MQSeries Everyplace アダプターが提供する必要のあるメソッドの定義です。新しいアダプターは、MQeAdapter から継承します。
MQeAdminMsg	管理メッセージの基本を提供します。
MQeAttribute	認証、暗号化、および圧縮を行うメカニズムが入っています。
MQeChannelListener	着信 MQSeries Everyplace 論理チャネルのリスナーを作成するために使用されます。
MQeChannelManager	MQSeries Everyplace 論理チャネルのマネージャーを作成します。
MQeEnumeration	MQSeries Everyplace メッセージ・オブジェクトのコレクションを保持します。
MQeException	MQeException オブジェクトを作成します。
MQeFields	データ項目を保持し、データのダンプおよびリストアを行うメカニズムを提供します。
MQeKey	属性オブジェクトに付加し、属性オブジェクトによって使用される MQeKey オブジェクトを作成します。
MQeMessageEvent	MQeMessage イベントが発生するときに、アプリケーションに渡される MQeMessageEvent オブジェクトを作成します。
MQeMsgObject	これは、データを保持するか、データを取得するまたはデータをある MQSeries Everyplace システムから別のシステムへ送信するために必要な論理を含んでいます。

表 1. com.ibm.mqe パッケージのクラス (続き)

クラス名	目的
MQeQueue	MQSeries Everyplace キュー・オブジェクトを作成します。
MQeQueueManager	MQSeries Everyplace キュー・マネージャー・オブジェクトを作成します。
MQeQueueManagerConfigure	キュー・マネージャーおよびデフォルトのキューを作成および削除するために使用されます。
MQeQueueManagerRules	キュー・マネージャーが特定の操作を実行するときに呼び出されるメソッドが入っています。
MQeQueueRule	キューで特定のイベントが発生するときに呼び出されるメソッドが入っています。
MQeRule	スーパークラスです。すべての MQSeries Everyplace ルール・クラスがこれから基本機能を派生します。

表 2. com.ibm.mqe パッケージのインターフェース

インターフェース名	目的
MQeEventLogInterface	すべての MQSeries Everyplace ログ・ハンドラーはこのインターフェースをインプリメントする必要があります。
MQeMessageListenerInterface	MQeMessage イベントを受け取る必要のあるすべてのオブジェクトは、このインターフェースをインプリメントする必要があります。
MQeRunListInterface	これにより、MQSeries Everyplace アプリケーションのリストが、キュー・マネージャーがアクティブになるときに渡されます。
MQeSecurityInterface	これは、 Java® セキュリティー・マネージャーが呼び出しを許可または拒否するためのオプション・インターフェースです。
MQeTraceInterface	すべての MQSeries Everyplace トレース・ハンドラーはこのインターフェースをインプリメントする必要があります。

com.ibm.mqe.administration

表 3. com.ibm.mqe.administration パッケージのクラス

クラス名	目的
MQeAdminQueueAdminMsg	MQeAdminQueue タイプのキューを管理するために使用されます。

表 3. *com.ibm.mqe.administration* パッケージのクラス *(続き)*

クラス名	目的
MQeConnectionAdminMsg	MQeConnectionDefinition タイプの接続を管理するために使用されるクラスです。
MQeHomeServerQueueAdminMsg	MQeHomeServerQueue タイプのキューを管理するために使用されます。
MQeQueueAdminMsg	MQeQueue タイプの MQSeries Everyplace ローカル・キューを管理するために使用されます。
MQeQueueManagerAdminMsg	MQeQueueManager タイプのキュー・マネージャーを管理するために使用されます。
MQeRemoteQueueAdminMsg	MQeRemoteQueue タイプのリモート・キューを管理するために使用されます。
MQeStoreAndForwardQueueAdminMsg	MQeStoreAndForwardQueue タイプのキューを管理するために使用されます。

com.ibm.mqe.attributes

表 4. *com.ibm.mqe.attributes* パッケージのクラス

クラス名	目的
****MQe3DESCryptor**	3DES 暗号化のメカニズムを提供します。
MQeDESCryptor	DES 暗号化のメカニズムを提供します。
MQeGenDH	ソリューション固有の MQeDHk クラス・オブジェクトを作成するときの基になる MQeDHk.java ファイルを作成します。
MQeLocalSecure	シンプルなローカル・セキュリティー・サービスを提供します。
MQeLZWCompressor	LZW 圧縮のメカニズムを提供します。
****MQeMARSCryptor**	MARS 暗号化のメカニズムを提供します。
MQeMAttribute	シンプルなメッセージ・レベルの保護を提供します。
****MQeMTrustAttribute**	拡張機能を持つメッセージ・レベルの保護を提供します。
****MQeRC4Cryptor**	RC4 暗号化のメカニズムを提供します。
****MQeRC6Cryptor**	RC6 暗号化のメカニズムを提供します。
MQeRleCompressor	Run Length エンコード圧縮のメカニズムを提供します。
****MQeWTLSCertAuthenticator**	最小限の証明認証のメカニズムを提供します。
MQeXORCryptor	XOR 暗号化のメカニズムを提供します。

com.ibm.mqe.registry

表 5. *com.ibm.mqe.registry パッケージのクラス*

クラス名	目的
MQePrivateRegistry	一連の専用オブジェクトおよび共通オブジェクトに対するアクセスを制御するための、専用レジストリー・オブジェクトを作成します。
MQePrivateRegistryConfigure	専用レジストリーを構成するために使用されます。
MQePublicRegistry	一連の共通オブジェクトに対するアクセスを制御するための、公開レジストリー・オブジェクトを作成します。

com.ibm.mqe.server

表 6. *com.ibm.mqe.server パッケージのインターフェース*

インターフェース名	目的
****MQeMiniCertIssuanceInterface**	MQeMiniCertificateServerGUI のインスタンスが新しい最小限の証明の発行を管理する方法を定義するときに使います。

com.ibm.mqe.mqemqmessage

表 7. *com.ibm.mqe.mqemqmessage パッケージのインターフェース*

インターフェース名	目的
MQeMQMsgObject	MQSeries Everyplace 内の MQSeries 形式のメッセージ・オブジェクトを表すのに使用されます。

com.ibm.mqe.mqbridge

表 8. *com.ibm.mqe.mqbridge パッケージのクラス*

クラス名	目的
MQeCharacteristicLabels	MQSeries-Bridge・コード内で使用する MQeFields オブジェクトのすべてが使用するラベル を 1 つのグループにまとめます。
MQeClientConnectionAdminMsg	MQeClientConnection オブジェクトに対して実行する管理コマンドをカプセル化するために使用します。

表 8. com.ibm.mqe.mqbridge パッケージのクラス (続き)

クラス名	目的
MQeListenerAdminMsg	MQeListener オブジェクトに対して実行する管理コマンドをカプセル化するために使用します。
MQeMQBridgeAdminMsg	MQSeries-Bridge・オブジェクトに対して実行する管理コマンドをカプセル化するために使用します。
MQeMQBridgeQueue	このキューは、MQSeries-Bridgeに対するインターフェースとして使用します。
MQeMQBridgeQueueAdminMsg	MQSeries-Bridge・キューを管理するために使用します。
MQeMQBridges	特定の MQSeries Everyplace サーバーに関連付けられているすべての MQSeries-Bridge・オブジェクトをロードして保守します。
MQeMQBridgesAdminMsg	MQeMQBridges オブジェクトに対して実行する管理コマンドをカプセル化するために使用します。
MQeMQQMgrProxyAdminMsg	MQeQMgrProxy オブジェクトに対して実行する管理コマンドをカプセル化するために使用します。
MQeRunState	管理対象オブジェクトの*実行状態* を保持します。

表 9. com.ibm.mqe.mqbridge パッケージのインターフェース

インターフェース名	目的
MQeTransformerInterface	MQMessage を MQeMsgObject に (またはその逆に) 変換できるすべてのクラスは、このインターフェースに準拠しなければなりません。

com.ibm.mqe.adapters

表 10. com.ibm.mqe.adapters パッケージのクラス

クラス名	目的
MQeDiskFieldsAdapter	ローカル・ディスクに対する MQeFields 情報の読み取りと書き込みをサポートします。
MQeMemoryFieldsAdapter	MQeFields 情報を一時的に保管します。
MQeReducedDiskFieldsAdapter	ローカル・ディスクへの MQeFields 情報の高速の書き込みをサポートします。

表 10. com.ibm.mqe.adapters パッケージのクラス (続き)

クラス名	目的
MQeTcpipAdapter	TCP/IP ストリームにおけるデータの読み取りと書き込みをサポートします。
MQeTcpipHttpAdapter	MQeTcpipAdapter の拡張クラスであり、HTTP 1.0 プロトコルの基本的なサポートを提供します。
MQeTcpipLengthAdapter	MQeTcpipAdapter の拡張クラスであり、単純でバイト効率のよいプロトコルを提供します。
MQeTcpipHistoryAdapter	MQeTcpipLengthAdapter の拡張クラスであり、最近使用したデータをキャッシュに入れるための効率のよいプロトコルを提供します。
MQeUdpipAdapter	UDP/IP データグラムへの保証付きデータ転送をサポートします。
MQeWESAuthenticationAdapter	Websphere Everyplace 認証プロキシーと透過的プロキシーによって、HTTP 要求のトンネリングをサポートします。

第2章 com.ibm.mqe のクラス

ここでは、 MQSeries Everyplace の以下のクラスとインターフェースについて詳しく説明します。

表 11. com.ibm.mqe パッケージのクラス

クラス名	目的
MQe	他の MQSeries Everyplace クラスを派生させます。
MQeAbstractMessageStore	メッセージ・ストレージ・インターフェースを定義します。
MQeAdapter	これは、すべての MQSeries Everyplace アダプターが提供する必要のあるメソッドの定義です。新しいアダプターは、MQeAdapter から継承します。
MQeAdminMsg	管理メッセージの基本を提供します。
MQeAttribute	認証、暗号化、および圧縮を行うメカニズムが入っています。
MQeChannelListener	着信 MQSeries Everyplace 論理チャネルのリスナーを作成するために使用されます。
MQeChannelManager	MQSeries Everyplace 論理チャネルのマネージャーを作成します。
MQeEnumeration	MQSeries Everyplace メッセージ・オブジェクトのコレクションを保持します。
MQeException	MQeException オブジェクトを作成します。
MQeFields	データ項目を保持し、データのダンプおよびリストアを行うメカニズムを提供します。
MQeKey	属性オブジェクトに付加し、属性オブジェクトによって使用される MQeKey オブジェクトを作成します。
MQeMessageEvent	MQeMessage イベントが発生するときに、アプリケーションに渡される MQeMessageEvent オブジェクトを作成します。
MQeMsgObject	これは、データを保持するか、データを取得するまたはデータをある MQSeries Everyplace システムから別のシステムへ送信するために必要な論理を含んでいます。
MQeQueue	MQSeries Everyplace キュー・オブジェクトを作成します。
MQeQueueManager	MQSeries Everyplace キュー・マネージャー・オブジェクトを作成します。
MQeQueueManagerConfigure	キュー・マネージャーおよびデフォルトのキューを作成および削除するために使用されます。

表 11. com.ibm.mqe パッケージのクラス (続き)

クラス名	目的
MQeQueueManagerRule	キュー・マネージャーが特定の操作を実行するときに呼び出されるメソッドが入っています。
MQeQueueRule	キューで特定のイベントが発生するときに呼び出されるメソッドが入っています。
MQeRule	スーパークラスです。すべての MQSeries Everyplace ルール・クラスがこれから基本機能を派生します。

表 12. com.ibm.mqe パッケージのインターフェース

インターフェース名	目的
MQeEventLogInterface	すべての MQSeries Everyplace ログ・ハンドラーはこのインターフェースをインプリメントする必要があります。
MQeMessageListenerInterface	MQeMessage イベントを受け取る必要のあるすべてのオブジェクトは、このインターフェースをインプリメントする必要があります。
MQeRunListInterface	これにより、MQSeries Everyplace アプリケーションのリストが、キュー・マネージャーがアクティブになるときに渡されます。
MQeSecurityInterface	これは、Java セキュリティー・マネージャーが呼び出しを許可または拒否するためのオプション・インターフェースです。
MQeTraceInterface	すべての MQSeries Everyplace トレース・ハンドラーはこのインターフェースをインプリメントする必要があります。

MQe

このクラスは、他の MQSeries Everyplace クラスを派生するために使用されます。これには、MQSeries Everyplace のプログラミングに役立つさまざまな定数の定義や、ユーティリティー・メソッドが入っています。通常の環境では、アプリケーション・クラスはこのクラスから継承します。たとえば、'class xxxxx extends MQe'。

パッケージ　　com.ibm.mqe

このクラスは、Object の下位クラスであり、 Serializable をインプリメントします。

定数

このクラスは以下に示す定数を提供します。

MQeMsgObject フィールド名

```
public  final static String   Msg_CorrelID
public  final static String   Msg_MsgID
public  final static String   Msg_OriginQMgr
public  final static String   Msg_Priority
public  final static String   Msg_Time
public  final static String   Msg_ReplyToQ
public  final static String   Msg_ReplyToQMgr
public  final static String   Msg_Style
public  final static String   Msg_LockID
public  final static String   Msg_Resend
public  final static String   Msg_ExpireTime
public  final static String   Msg_WrapMsg
```

メッセージ・スタイルの修飾子

```
public  final static int      Msg_Style_Datagram
public  final static int      Msg_Style_Request
public  final static int      Msg_Style_Reply
```

標準のキュー名

```
public  final static String   Admin_Queue_Name
public  final static String   Admin_Reply_Queue_Name
public  final static String   DeadLetter_Queue_Name
public  final static String   System_Default_Queue_Name
```

MQeAdapter オブジェクトと使用するためのオプション

```
public  final static String   MQe_Adapter_APPEND
public  final static String   MQe_Adapter_BINARY
public  final static String   MQe_Adapter_CONTENT
public  final static String   MQe_Adapter_FINAL
public  final static String   MQe_Adapter_FLUSH
public  final static String   MQe_Adapter_HEADER
public  final static String   MQe_Adapter_HEADERRSP
public  final static String   MQe_Adapter_LENGTH
```

```
public  final static String   MQe_Adapter_LISTEN
public  final static String   MQe_Adapter_PERSIST
public  final static String   MQe_Adapter_READ
public  final static String   MQe_Adapter_RESET
public  final static String   MQe_Adapter_SYNC
public  final static String   MQe_Adapter_UNICODE
public  final static String   MQe_Adapter_UPDATE
public  final static String   MQe_Adapter_WRITE
```

MQeAdapter オブジェクトと使用するための制御オプション

```
public  final static String   MQe_Adapter_ACCEPT
public  final static String   MQe_Adapter_FILENAME
public  final static String   MQe_Adapter_FILTER
public  final static String   MQe_Adapter_GETPERSIST
public  final static String   MQe_Adapter_LIST
public  final static String   MQe_Adapter_PULSE
public  final static String   MQe_Adapter_QOSINPUTS
public  final static String   MQe_Adapter_SECTION
public  final static String   MQe_Adapter_SETSOCKET
```

MQeAdapter オブジェクトと使用するための状況オプション

```
public  final static String   MQe_Adapter_BYTECOUNTS
public  final static String   MQe_Adapter_LOCALHOST
public  final static String   MQe_Adapter_LINKPARM
public  final static String   MQe_Adapter_NETWORK
```

サービス品質フィールド名

```
public  final static String   QoS_BytesRead
public  final static String   QoS_BytesWritten
public  final static String   QoS_Cost
public  final static String   QoS_DialRetry
public  final static String   QoS_DialRetryWait
public  final static String   QoS_Duration
public  final static String   QoS_ErrorRate
public  final static String   QoS_Errors
public  final static String   QoS_Jitter
public  final static String   QoS_Latency
public  final static String   QoS_Pulse
public  final static String   QoS_Rate
public  final static String   QoS_Retry
public  final static String   QoS_Size
public  final static String   QoS_TimeOut
```

ログ・インターフェースのログ・タイプ

```
public  final static byte     MQe_Log_SUCCESS
public  final static byte     MQe_Log_ERROR
public  final static byte     MQe_Log_WARNING
public  final static byte     MQe_Log_INFORMATION
```

例外索引番号

```
  public  final static int      Except_UnCoded
  public  final static int      Except_Debug
  public  final static int      Except_NotSupported
  public  final static int      Except_Syntax
  public  final static int      Except_Type
  public  final static int      Except_Command
  public  final static int      Except_NotFound
  public  final static int      Except_Data
  public  final static int      Except_BadRequest
  public  final static int      Except_Stopped
  public  final static int      Except_Closed
  public  final static int      Except_Duplicate
  public  final static int      Except_NotAllowed
  public  final static int      Except_Rule
  public  final static int      Except_TimeOut

  public  final static int      Except_InvalidHandle
  public  final static int      Except_AllocationFail

  public  final static int      Except_Chnl_Attributes
  public  final static int      Except_Chnl_Destination
  public  final static int      Except_Chnl_Limit
  public  final static int      Except_Chnl_ID
  public  final static int      Except_Chnl_Overrun

  public  final static int      Except_Trnsport_QMgr
  public  final static int      Except_Trnsport_Request

  public  final static int      Except_QMgr_NotActive
  public  final static int      Except_QMgr_InvalidQMgrName
  public  final static int      Except_QMgr_Activated
  public  final static int      Except_QMgr_AlreadyExists
  public  final static int      Except_QMgr_InvalidQName
  public  final static int      Except_QMgr_QExists
  public  final static int      Except_QMgr_UnknownQMgr
  public  final static int      Except_QMgr_QNotEmpty
  public  final static int      Except_QMgr_QDoesNotExist
  public  final static int      Except_QMgr_QInUse
  public  final static int      Except_QMgr_WrongQType
  public  final static int      Except_QMgr_InvalidChannel
  public  final static int      Except_QMgr_SecureMsgDecodeFailed
  public  final static int      Except_QMgr_NotConfigured
  public  final static int      Except_QMgr_Busy

  public  final static int      Except_Q_NoMsgAvailable
  public  final static int      Except_Q_NoMatchingMsg
  public  final static int      Except_Q_InvalidPriority
  public  final static int      Except_Q_Full
  public  final static int      Except_Q_MsgTooLarge
  public  final static int      Except_Q_NotActive
  public  final static int      Except_Q_Active
  public  final static int      Except_Q_InvalidName
  public  final static int      Except_Q_TargetRegistryRequired

public  final static int      Except_Uncontactable_DontTransmit
```

```
public  final static int      Except_RasDialFailed
public  final static int      Except_RasGetProjectionInfoFailed
public  final static int      Except_RasHangUpFailed

public  final static int      Except_Connect_AdapterNotActive
public  final static int      Except_Connect_InvalidDefinition

public  final static int      Except_Con_AlreadyExists
public  final static int      Except_Con_AliasAlreadyExists
public  final static int      Except_Con_AdapterRequired
public  final static int      Except_Con_InvalidName
public  final static int      Except_Client_Con_Not_Available

public  final static int      Except_Reg_NullName
public  final static int      Except_Reg_AlreadyExists
public  final static int      Except_Reg_DoesNotExist
public  final static int      Except_Reg_OpenFailed
public  final static int      Except_Reg_InvalidSession
public  final static int      Except_Reg_NotDefined
public  final static int      Except_Reg_AddFailed
public  final static int      Except_Reg_DeleteFailed
public  final static int      Except_Reg_ReadFailed
public  final static int      Except_Reg_UpdateFailed
public  final static int      Except_Reg_ListFailed
public  final static int      Except_Reg_SearchFailed
public  final static int      Except_Reg_RenameFailed
public  final static int      Except_Reg_ResetPINFailed
public  final static int      Except_Reg_CRTKeyDecFailed
public  final static int      Except_Reg_CRTKeySignFailed
public  final static int      Except_Reg_DeleteRegistryFailed
public  final static int      Except_Reg_AlreadyOpen
public  final static int      Except_Reg_NotSecure

public  final static int      Except_PrivateReg_BadPIN
public  final static int      Except_PrivateReg_ActivateFailed
public  final static int      Except_PrivateReg_NotOpen

public  final static int      Except_MiniCertReg_BadPIN
public  final static int      Except_MiniCertReg_ActivateFailed
public  final static int      Except_MiniCertReg_NotOpen

public  final static int      Except_PublicReg_ActivateFailed
public  final static int      Except_PublicReg_InvalidRequest

public  final static int      Except_Admin_NotAdminMsg
public  final static int      Except_Admin_ActionNotSupported
public  final static int      Except_Admin_InvalidField

public  final static int      Except_Authenticate
public  final static int      Except_S_Cipher
public  final static int      Except_S_InvalidSignature
public  final static int      Except_S_CertificateExpired
public  final static int      Except_S_InvalidAttribute
public  final static int      Except_S_MiniCertNotAvailable
public  final static int      Except_S_RegistryNotAvailable
public  final static int      Except_S_BadIntegrity
public  final static int      Except_S_NoPresetKeyAvailable
public  final static int      Except_S_MissingSection
```

ログ・レコード・タイプ

```
public  final static byte     MQe_Log_Success
public  final static byte     MQe_Log_Error
public  final static byte     MQe_Log_Warning
public  final static byte     MQe_Log_Information
public  final static byte     MQe_Log_Audit_Success
public  final static byte     MQe_Log_Audit_Failure
```

イベント

```
public  final static int      Event_Activate
public  final static int      Event_Close
public  final static int      Event_Logon
public  final static int      Event_Logoff
public  final static int      Event_QueueManager
public  final static int      Event_Queue
public  final static int      Event_Attribute
public  final static int      Event_Authenticate
public  final static int      Event_MiniCert_Validate
public  final static int      Event_UserBase
```

公的にアクセス可能な変数

debugCall:

true にセットされると、スタック・トレースおよび MQeFields オブジェクトの内容が System.err.println に書き込まれます。

```
public static boolean debugCall  = false;
```

debugExcept:

true にセットされると、MQSeries Everyplace システム内で特定の例外が発生するときに、スタック・トレースが System.err.println に印刷されます (これらの例外は try ... catch ... によって処理され、通常は表示されません)。

```
public static boolean debugExcept  = false;
```

debugMQeExcept:

true にセットされると、 MQeException の発生ごとに、スタック・トレースが System.err.println に印刷されます。

```
public static boolean debugMQeExcept  = false;
```

loader: これは、クラス・ファイルをローカル・システムまたはリモート・システムから動的にロードできるようにするクラス・ローダーへのオブジェクト参照です。

```
public static MQeLoader loader
```

MQeObjectCount:

MQe クラスの下位である、インスタンス化されたオブジェクトの現在の数を含む整数です。

```
public static int MQeObjectCount
```

メソッドの要約

静的メソッド

メソッド	目的
abbreviate	提供されたクラス名の完全名または省略名を戻します。
alias	クラス名の別名を追加します。
asciiToByte	ASCII ストリングをバイト配列に変換します。
byteToAscii	バイト配列を ASCII ストリングに変換します。
byteToHex	バイト配列を ASCII 16 進数に変換します。
byteToInt	4 バイトを int 値に変換します。
byteToLong	8 バイトを long 値に変換します。
byteToShort	2 バイトを short 値に変換します。
byteToUnicode	バイト配列を Unicode ストリングに変換します。
debug	現在の呼び出しスタックを `System.err.println` に印刷します。これはデバッグのために使用されます。
getEventLogHandler	現在のアクティブ・ログ・ハンドラーまたはヌル への参照を戻します。
getTraceHandler	現在のアクティブ・トレース・ハンドラーまたはヌル への参照を戻します。
hexToByte	ASCII 16 進数をバイト配列に変換します。
intToByte	int 値を 4 バイトに変換します。
log	ログ・ハンドラーにデータをロードします。
mapFileDescriptor	ファイル記述子にストリングをマップします。
setEventLogHandler	ログ要求を処理するクラスを設定します。
SetTraceHandler	トレース・メッセージを処理するクラスを設定します。
sliceByteArray	スライスをバイト配列以外のものにコピーします。
unicodeToByte	Unicode ストリングをバイト配列に変換します。
unicodeToUTF	Unicode ストリングを UTF エンコード・バイト配列に変換します。
uniqueValue	現在の環境に固有の long 値を生成します。
utfToUnicode	UTF エンコード・バイト配列を Unicode ストリングに変換します。

非静的メソッド

メソッド	目的
trace	トレース・メッセージを、 System.out.println または System.err.println に書き込みます。
type	クラス名のストリング表現を戻します。

MQe abbreviate

構文

```
public static String abbreviate(String className, int index )
```

説明 このメソッドは、省略形のクラス名を決定するか、クラス名を省略します。

省略されたクラス名は、"n:" の形式になります。ここで n は MQSeries Everyplace クラス名を表現する数値です。

たとえば、"6" は "com.ibm.mqe.MQeTransporter" になります。

パラメーター

className クラス名または省略形のクラス名の入ったストリング。

index 整数。現在サポートされている値は以下のとおりです。

0 省略名を完全修飾クラス名に変換します。

1 完全修飾名を省略形に変換します。

戻り値 完全修飾クラス名か省略名のストリングです。

例外

java.lang.ArrayOutOfBounds 無効な索引を指定しました。

例

```
class MyApplication
  {
  ...
  ...
  String abbrev = MQe.abbreviate( "com.ibm.mqe.MQeTransporter", 1 );
  ...
  }
```

MQe alias

構文

```
public static void alias( String from, String to )
```

説明 このメソッドは、クラスの別名を追加または除去します。 *from* パラメーターは別名であり、 *to* パラメーターは完全クラス名です。別名を除去するには、*to* パラメーターを ヌルに設定します。

パラメーター

from 別名の入ったストリングです。

to この別名の完全クラス名が入っているか、別名を除去するためにヌルであるストリングです。

戻り値 なし

例外 なし

例

```
class MyApplication
   {
   ...
   ...
   MQe.alias( "Network", "com.ibm.MQe.Adapters.MQeTcpipHttpAdapater" );
   ...
   }
```

MQe asciiToByte

構文

```
public static byte[] asciiToByte( String data )
```

説明 このメソッドは、ストリングを各文字の下位バイトだけを保持するバイト配列に変換します。

パラメーター

data ASCII データの入ったストリングです。

戻り値 ASCII データの入ったバイト配列です。

例外 なし

例

```
class MyApplication
  {
  ...
  ...
  byte data[] = MQe.asciiToByte( "This is some test data" );
  ...
  }
```

MQe byteToAscii

構文

```
public static String byteToAscii( byte data[] )
```

説明 このメソッドは、バイトをストリングの各文字の下位バイトにコピーすることによって、バイト配列を ASCII ストリングに変換します。

パラメーター

data 変換されるデータの入ったバイト配列です。

戻り値 変換されたデータの入ったストリングです。

例外 なし

例

```
class MyApplication
  {
  ...
  ...
    String data = MQe.byteToAscii( new byte[] { 64, 65, 66, 67, 68 } );
  ...
  }
```

MQe byteToHex

構文

```
public static String byteToHex( byte data )
public static String byteToHex( byte data[], int offset, int count )
```

説明 このメソッドは、バイト配列を、データの 16 進表記文字の入ったストリングに変換します。

パラメーター

data 変換されるデータの入ったバイト配列です。

offset データ配列内のスタート・エレメント索引です。

count 変換されるエレメントの数です。

戻り値 16 進ストリングです。

例外 なし

例

```
  ...
  String hexData = byteToHex( ByteArray );
  ...
```

MQe byteToInt

構文

```
public static int byteToInt( byte data[], int offset )
```

説明 このメソッドはバイト配列を整数値に変換します。

パラメーター

data 変換されるデータの入ったバイト配列です。

offset データ配列内のスタート・エレメント索引です。

戻り値 16 進ストリングです。

例外 なし

例

```
...
int value = byteToInt( ByteArray );
...
```

MQe byteToLong

構文

```
public static int byteToLong( byte data[], int offset )
```

説明 このメソッドはバイト配列を整数値に変換します。

パラメーター

data 変換されるデータの入ったバイト配列です。

offset データ配列内のスタート・エレメント索引です。

戻り値 long 整数値です。

例外 なし

例

```
...
 long value = byteToLong( byteArray, 0 );
...
```

MQe byteToShort

構文

```
public static int byteToShort( byte data[], int offset )
```

説明 このメソッドはバイト配列を short 整数値に変換します。

パラメーター

data 変換されるデータの入ったバイト配列です。

offset データ配列内のスタート・エレメント索引です。

戻り値 short 整数値です。

例外 なし

例

```
...
 short value = byteToShort( byteArray, 0 );
...
```

MQe byteToUnicode

構文

```
public static String byteToUnicode( byte data[] )
```

説明 このメソッドはバイト配列を Unicode ストリングに変換します。

パラメーター

data 変換されるデータの入ったバイト配列です。

戻り値 Unicode ストリングです。

例外 なし

例

```
...
 String data = byteToUnicode( ByteArray );
...
```

MQe debug

構文

```
public static void debug( String data )
```

説明 スタック・トレースを System.err.println に書き込むようにします。その後に、ストリング・データを続けます。処理は通常どおり継続されます。

パラメーター

data このスタック出力を識別するためのデータの入ったストリングです。

戻り値 なし

例外 なし

例

```
class MySampleClass extends MQe
  {   MySampleClass ( )
     {
     ...
     MQe.debug( "" );
     ...
     }
  ...
  }
```

MQe getEventLogHandler

構文

```
Public static MQeEventLogInterface getEventLogHandler( )
```

説明 現在のイベント・ログ・ハンドラー・オブジェクトを戻します。

パラメーター

なし

戻り値 ログ・ハンドラー・オブジェクトまたはヌルです。

例外 なし

例

```
class MySampleClass extends MQe
   {
  MySampleClass ( )
     {
     ...
     MQeEventLogInterface Logger= MQe.getEventLogHandler( );
     ...
      }
   ...
   }
```

MQe getTraceHandler

構文

```
Public static MQeTraceInterface getTraceHandler( )
```

説明 現在のトレース・ハンドラー・オブジェクトを戻します。

パラメーター

なし

戻り値 トレース・ハンドラー・オブジェクトまたはヌルです。

例外 なし

例

```
class MySampleClass extends MQe
   {
  MySampleClass ( )
     {
     ...
     MQeTraceInterface Logger= MQe.getTraceHandler( );
     ...
      }
  ...
   }
```

MQe hexToAscii

構文

```
public static String hexToAscii( String data ) throws Exception
```

説明 このメソッドは、データの 16 進表記文字の入ったストリングをバイト配列に変換します。

パラメーター

data 変換されるデータの入ったストリングです。

戻り値 変換されたデータの入ったストリングです。

例外 なし

例

```
...
 String data = hexToAscii( "30313233343536373839" );
...
```

MQe hexToByte

構文

```
public static byte[] hexToByte( String data ) throws Exception
```

説明 このメソッドは、データの 16 進表記文字の入ったストリングをバイト配列に変換します。

パラメーター

data 変換されるデータの入ったストリングです。

戻り値 変換されたデータの入ったバイト変換です。

例外 なし

例

```
...
 byte data[] = hexToByte( "30313233343536373839" );
...
```

MQe intToByte

構文

```
public static byte[] intToByte( int data )
```

説明 整数値を 4 バイトのバイト配列に変換します。

パラメーター

data 変換されるデータの入った整数です。

戻り値 変換されたデータの入ったバイト配列です。

例外 なし

例

```
 ...
 byte data[] = intToByte( "30313233343536373839" );
...
```

MQe log

構文

```
public static void log( byte logType, int logNumber, Object logData)
```

説明 メッセージをイベント・ログ・ルーチンに送ります。

パラメーター

logType ログ・メッセージのタイプの入ったバイトです。たとえば、以下のとおりです。

- MQe.MQe_Log_Success
- MQe.MQe_Log_Error
- MQe.MQe_Log_Warning
- MQe.MQe_Log_Information
- MQe.MQe_Log_Audit_Success
- MQe.MQe_Log_Audit_Failure

logNumber メッセージを識別する整数です。

logData ログに記録されるメッセージ・データの入ったストリングです。

戻り値 なし

例外 なし

例

```
...
  try
    {
    setLogHandler( new MyLogHandler( ... );
    log( MQe.MQE_LogSuccess, 123, "TEST opened" );
    ...
    }
  catch ( Exception e )
    {
    log( MQe.MQE_LogError, 123, "TEST failed" );
    }
  ...
```

MQe mapFileDescriptor

構文

```
public static void mapFileDescriptor(  String filedDesc,
                                       Object newDesc[] )
```

説明 別名またはニックネームをファイル記述子、パラメーター、およびオプションに割り当てます。このメッセージは通常内部で使用されます。

パラメーター

fileDesc ファイル記述子の入ったストリングです。

newDesc 新しいファイル記述子、およびパラメーターとオプション・データの入ったオブジェクト配列です。

戻り値 なし

例外 なし

例

```
...
MQe.MapFileDescriptor( "QMgrName", new String[] {
  "TcpipHttp:127.0.0.1:8080",
  "?Channel",
                        "" } );
...
...
```

MQe setEventLogHandler

構文

```
public static MQeEventLogInterface setEventLogHandler(
                                MQeLogInterface logObj )
```

説明 現在のイベント・ログ・ハンドラー・オブジェクトを戻し、MQSeries Everyplace が使用できるように新しいハンドラーを設定します。ログ・ハンドラー・オブジェクトはすべてのログ要求上での制御を取得します。

パラメーター

logObj MQeEventLogInterface に準拠したログ・ハンドラー・オブジェクトです。

戻り値 以前のログ・ハンドラー・オブジェクトまたはヌルです。

例外 なし

例

```
class MySampleClass extends MQe
   {
  MySampleClass ( )
     {
     super( );
     setEventLogHandler( new Examples.Log.LogToDiskFile( "ThisFile.log" ) );
     ...
     }
   ...
   }
```

MQe setTraceHandler

構文

```
public static MQeTraceInterface setTraceHandler( MQeTraceInterface traceObj )
```

説明 現在のトレース・ハンドラー・オブジェクトを戻し、MQSeries Everyplace が使用できるように新しいハンドラーを設定します。トレース・ハンドル・オブジェクトはすべての **trace()** メソッド呼び出しに対する制御を取得します。

パラメーター

traceObj MQeTraceInterface に準拠したトレース・ハンドラー・オブジェクトです。

戻り値 以前のトレース・ハンドラー・オブジェクトまたはヌルです。

例外 なし

例

```
class MySampleClass extends MQe
   {
  MySampleClass ( )
     {
     super( );
     setTraceHandler( new MQeTraceWindow( "Window Title", null ) );
     ...
      }
   ...
   }
```

MQe sliceByteArray

構文

```
public static byte[] sliceByteArray( byte data[],
int offset,
int length )
```

説明 このメソッドは、*data [Offset]* で始まるデータから成り、エレメント数の *length* を持つバイトの配列を戻します。これは、*data* 配列の一部のコピーです。

パラメーター

data ソースのバイト配列です。

offset コピーされる *data* 内のスタート・エレメントです。

length コピーされるバイト数です。

戻り値 データからのエレメントのコピーの入ったバイト配列です。

例外 なし

例

```
class MySampleClass extends MQe
  {
  MySampleClass ( )
    {
    ...
    byte data[] = { (byte) 1, (byte) 2, (byte) 3, (byte) 4, (byte) 5, };

    byte temp[] = sliceByteArray( data, 1, 3 );
    ...
    }
  ...
  }
```

MQe trace

構文

```
public void trace( String msg )

public void trace(int msgNumber, long insert)

public void trace( int msgNumber, Object insert )
```

説明 メッセージをトレース・ルーチンに送ります。

パラメーター

msg プレフィックス文字およびメッセージの入ったストリングです。プレフィックス・バイトには以下が含まれます。

″ ″	ユーザー・メッセージ
″**I**″	通知メッセージ
″**W**″	警告メッセージ
″**E**″	エラー・メッセージ
″**S**″	セキュリティー・メッセージ
″**D**″	デバッグ・メッセージ
″_″	ユーザー・メッセージ
″**i**″	通知メッセージ
″**w**″	警告メッセージ
″**e**″	エラー・メッセージ
″**s**″	セキュリティー・メッセージ
″**d**″	デバッグ・メッセージ

msgNumber 表示されるメッセージの数の入った整数です。メッセージは MQeTrace.addMessage メソッドを使用して事前に追加されている必要があります。メッセージ番号は 0 <= msgNumber <= 32767 の範囲でなければなりません。

insert 整数値、オブジェクト・タイプ・ストリング、または String[] です。これは、メッセージ・テンプレートの insert ID の位置に挿入されます (詳細については、**trace()** の例を参照してください)。

戻り値 なし

例外

IOException 入出力エラーが発生

例

```
 ...
     {
     ...
     trace( "I:Information message" );
     trace( 5, "Error message text" );
     ...
     }
   ...
```

MQe type

構文

```
public String type( )
```

説明 オブジェクトのストリング名を戻します。

注: これによって、省略形のクラス名が戻される場合とそうでない場合があります。 **abbreviate()** メソッドを参照してください。

パラメーター

なし

戻り値 オブジェクト名の入ったストリングです。

例外 なし

例

```
...
MQe.MQeMsgObject object = new MQeMsgObject( );
String objectName = object.type();
...
```

MQe unicodeToByte

構文

```
public static byte[] unicodeToByte( String data )
```

説明 このメソッドは、Unicode ストリングをバイト配列に変換します。

パラメーター

data 変換されるデータの入ったバイト配列です。

戻り値 変換されたデータの入ったバイト変換です。

例外 なし

例

```
...
byte data[] = unicodeToByte( "This is a Data string" );
...
```

MQe unicodeToUTF

構文

```
public static byte[] unicodeToUTF( String data )
```

説明 このメソッドは、Unicode ストリングをバイト配列に変換します。

パラメーター

data 変換されるデータの入ったバイト配列です。

戻り値 変換されたデータの入ったバイト配列です。

例外 なし

例

```
...
byte data[] = unicodeToUTF( "This is a Data string" );
...
```

MQe uniqueValue

構文

```
public long uniqueValue( )
```

説明 現在の JVM 内で固有となる long 値を戻します。

パラメーター

なし

戻り値 現在の環境に固有の long 整数値です。

例外 なし

例

```
...
long number = MQe.uniqueValue( );
...
```

MQe utfToUnicode

構文

```
public static String utfToUnicode( byte data[])
```

説明 このメソッドは、 UTF エンコード Unicode の入ったバイト配列を Unicode ストリングに変換します。

パラメーター

data 変換されるデータの入ったバイト配列です。

戻り値 Unicode ストリングです。

例外 なし

例

```
...
 String data = MQe.utfToUnicode( byteArray );
...
```

MQeAbstractMessageStore

このクラスは、メッセージ・ストレージ・インターフェースを定義します。以下を含む MQSeries Everyplace キューのメッセージ・ストレージと検索アクティビティーをサポートするために設計されています。

- メッセージを保管する
- フィルターに掛けてメッセージを検索する
- 何も知らせずにメッセージを期限切れにする
- メッセージを削除する
- 書き込みと取得の確認を管理する
- メッセージのロックを管理する
- メッセージをブラウズする
- メッセージの属性が圧縮またはセキュリティーである場合、効率的または安全にラッピングする

効率のよいストレージと検索のアルゴリズムのインプリメントは、サブクラスで実行します。

デフォルトのインプリメンテーションでは、以下のメッセージ・フィールドを索引で使用します。

- `MQe.Msg_OriginQMgr`
- `MQe.Msg_Time`
- `MQe.Msg_MsgID`
- `MQe.Msg_CorrelID`
- `MQe.Msg_Priority`
- `MQe.Msg_ExpireTime`
- `MQe.Msg_LockID`
- `MSG_DESTINATION_QUEUEMANAGER`
- `MSG_DESTINATION_QUEUE`

サブクラスをインプリメントする際には、索引で使用するフィールドを自由に選択できます (索引を使用する場合)。

このクラスのメソッドは、抽象または最終のどちらにでも宣言されます。抽象メソッドは、サブクラスによるインプリメンテーションを必要とします。メッセージ・ストアをインプリメントする際には、必要に応じて自由にインプリメントできますが、その動作が、所有するキューによって決まっているものもあります。期待される動作を最も適確に定義するには、サブクラスの例 MQeMessageStore を使用します。

パッケージ　　com.ibm.mqe

このクラスは、MQe の拡張クラスです。

コンストラクター

MQeAbstractMessageStore

構文

```
public MQeAbstractMessageStore()
```

説明

パラメーター

なし

戻り値 なし

例外 なし

メソッド

抽象メソッド

メソッド	目的
browseMessages	ロック付きのメッセージをブラウズします。
close	メッセージの保管を非活動化します。
confirm	メッセージを確認します。
confirmGetmessage	メッセージの正常な受信を確認します。
confirmPutMessage	書き込み操作を確認します。
deleteMessage	メッセージをキューから削除します。
getDefaultPriority	キューのデフォルト優先順位値を取得します。
getExpiryInterval	キューの有効期限間隔を取得します。
getMessage	メッセージをキューから取得します。
getNumberOfMessages	キューに現在あるメッセージ数を取得します。
getPendingMessage	キュー・マネージャーにより保留になっているメッセージを取得します。
open	メッセージ・ストアをオープンします。
putMessage	メッセージをキューに入れます。
resetLockedMsgs	すべてのロックされたメッセージをリセットします。
transmitAll	すべての保留メッセージを送信します。
transmitConfirmMessage	確認付きでメッセージを送信します。
transmitMessage	メッセージを送信します。
undo	操作が失敗した後、メッセージをその直前の状態にリセットします。
unlockMessage	以前にロックされたメッセージをアンロックします。
updateRetryCount	ロックされたメッセージの再試行数を更新します。
wrapMessage	メッセージをラップしてエンコードします。

最終メソッド

メソッド	目的
getStringArgument	キューの保管ロケーションを取得します。
rule	キュー・ルール・オブジェクトを取得します。
trace	単純化されたトレースを提供します。
unWrapMsg	エンコードされたメッセージをアンラップして、デコードします。
WrapMessage	メッセージをラップしてエンコードします。

MQeAbstractMessageStore browseMessages

構文

```
protected abstract Enumeration _browseMessages(MQeFields filter,
                                               MQeAttribute attribute,
                                               long confirmID,
                                               long lockID,
                                               boolean justUID) throws Exception
```

説明 このメソッドは、現在ロックされていないメッセージを戻します。 *confirmID* が非ゼロの場合、メッセージがロックされていることが検出されます。

パラメーター

filter MQeFields オブジェクト・メッセージ・フィルター。これには、操作を成功させるためのメッセージ固有の ID が含まれなければなりません。

attribute 復元されるデータを含むバイト配列をデコードするのに使用する MQeAdminQueueAdminMsg オブジェクト。

confirmID 確認 ID として使用される値。

lockID ロック ID として使用される値。

justUID メッセージ全体を戻すか、またはメッセージの *UID* (固有 ID) のみを戻すかを判断するのに使用されるブール値。

戻り値 フィルターに一致するメッセージ (または *UID*) を含む列挙型オブジェクト。

MQeAbstractMessageStore close

構文

```
protected abstract void close(long confirmID) throws Exception
```

説明 メッセージ・ストアのライフ・サイクルを非活動化し、終了させます。

パラメーター

なし

戻り値 なし

MQeAbstractMessageStore confirm

構文

```
protected abstract void confirm(long confirmID) throws Exception
```

説明 特定の *confirmID* を持つメッセージすべてを検査し、それを確認します。このメソッドは、取得と書き込みを確認し、さらにブラウズするためにロックされたメッセージをアンロックします。

パラメーター

confirmID confirmID を表す値。

戻り値 なし

例外 なし

MQeAbstractMessageStore confirmGetMessage

構文

```
protected abstract long confirmGetMessage(MQeFields filter) throws Exception
```

説明 メッセージ・ストアから直前の **getMessage()** 操作によって取得されたメッセージが、正常に受信されたことを確認します。メッセージは、確認フローが受信されるまで、メッセージ・ストアでロックされたままです。有効期限切れのメッセージは戻されません。

パラメーター

filter MQeFields オブジェクト・メッセージ・フィルター。操作が成功するためには、フィルターにメッセージ固有の ID が含まれていなければなりません。

戻り値 long 値のメッセージ時刻。トレースで使用できます。

例外

MQeException `Except_NotFound`

この例外は、すでに確認されたメッセージを確認しようとする場合に throw されます。アプリケーションがメッセージ取得要求の確認を再発行した場合、この例外は成功を示す戻りコードとして扱われます。

MQeAbstractMessageStore confirmPutMessage

構文

```
protected abstract MQeMsgObject confirmPutMessage(MQeFields filter,
                                                  boolean returnMessage) throws Exception
```

説明 書き込み操作を確認します。

パラメーター

filter メッセージ・フィルターを含む MQeFields オブジェクト。操作が成功するためには、フィルターにメッセージ固有の ID が含まれていなければなりません。

returnMessage ブール値。`true` の場合、呼び出し側はアンロックされたメッセージが戻されることを希望します。 `false` の場合は、メッセージは戻される必要はありません。メッセージの取得が、単純なアンロックよりコストのかかるメッセージ・ストアでは、これを使用してパフォーマンスを向上させることができます。

戻り値 戻されたメッセージの入っている MQeMsgObject。

例外

MQeException `Except_NotFound`

この例外が発生するのは、すでに確認されたメッセージを確認しようとしたときです。アプリケーションがメッセージ書き込み要求の確認を再発行した場合、この例外は成功を示す戻りコードとして扱われます。

MQeAbstractMessageStore deleteMessage

構文

```
protected abstract long deleteMessage(MQeFields filter) throws Exception
```

説明 このメソッドは、メッセージをメッセージ・ストアから削除します。 1 回の操作で削除できるメッセージは 1 つだけであり、メッセージ固有の ID が必要です。以前の操作でロックされたメッセージは、メッセージ・フィルターに有効な *lockID* を含めることによって削除できます。メッセージが使用可能でない場合、例外が戻されます。

パラメーター

filter MQeFields オブジェクト・メッセージ・フィルター。操作が成功するためには、フィルターにメッセージの *UID* が含まれていなければなりません。

戻り値 long 値のメッセージ時刻。トレースで使用できます。

例外

MQeException `Except_NotFound`

`Except_NotAllowed`

MQe AbstractMessageStore getDefaultPriority

構文

```
public final byte getDefaultPriority()
```

説明 メッセージ・ストアを所有するキューのデフォルト優先順位値を取得します。

パラメーター

なし

戻り値 キュー優先順位。

例外 なし

MQeAbstractMessageStore getExpiryInterval

構文

```
public final long getExpiryInterval()
```

説明 メッセージ・ストアを所有するキューの有効期限間隔を取得します。

パラメーター

なし

戻り値 メッセージ・ストアを所有するキューの有効期限間隔。

例外 なし

MQeAbstractMessageStore getMessage

構文

```
protected abstract MQeMsgObject getMessage(MQeFields filter,
                                           MQeAttribute attribute,
                                           long confirmID ) throws Exception;
```

説明 このメソッドは、指定されたメッセージ・ストアから使用可能なメッセージを戻します。そのメッセージは、メッセージ・ストアから除去されます。メッセージ・フィルターが指定されない場合、メッセージ・ストアで最初に使用可能なメッセージが戻されます。メッセージ・フィルターが指定されている場合、フィルターに一致する最初に使用可能なメッセージが戻されます。以前のブラウズ操作でロックされたメッセージは、メッセージのロックに使用された *lockID* をメッセージ・フィルターに含めることによって検索できます。有効期限切れのメッセージは戻されません。メッセージが使用できない場合、例外が throw されます。

保証されたメッセージ送達は、*confirmId* パラメーターの値に依存します。非ゼロ値を渡すと通常のメッセージが戻されますが、そのメッセージは、後続の確認が受け取られるまでロックされており、メッセージ・ストアから除去されません。確認は、**confirmGetMessage()** メソッドによって発行できます。ゼロの値を渡すと、メッセージが戻され、そのメッセージはターゲットのメッセージ・ストアから除去されますが、メッセージ送達は保証されません。*confirmId* パラメーターは、このコマンドの実行時にエラーが生じた場合にも使用されます。障害が生じうるのは、メッセージがアプリケーションに戻されておらず、メッセージがまだターゲットのメッセージ・ストアでロック状態になっている場合です。取得操作で使用したのと同じ *confirmID* を **undo()** メ

ソッドに渡せば、メッセージは以前の状態に戻ります。取得操作のたびに、固有の値を使用するようお勧めします。固有の値は、**MQe.uniqueValue()** メソッドを使用して生成できます。

パラメーター

filter ヌル、またはメッセージ・フィルターを含む MQeFields オブジェクト。

attribute メッセージ・レベルのセキュリティーを提供するために使用する MQeAdminQueueAdminMsg オブジェクト。提供される属性は、このメソッドで戻されるメッセージに付加された属性と一致しなければなりません。この属性が一致しない場合は、メッセージが逸失します。

confirmId 保証されたメッセージ送達を使用するかどうかを指示する long 値。非ゼロの場合、メッセージがメッセージ・ストアから除去されます。

戻り値 指定されたキューから取得されたメッセージを含む MQeMsgObject。

例外

MQeException `Except_Q_NoMatchingMsg`

`Except_NotFound`

MQeAbstractMessageStore getNumberOfMessages

構文

```
protected abstract int getNumberOfMessages()  throws Exception
```

説明 メッセージ・ストアに現在あるメッセージ数を取得します。

パラメーター

なし

戻り値 メッセージ数を含む値。

MQeAbstractMessageStore getPendingMessage

構文

```
protected abstract MQeMsgObject getPendingMessage(String queueManagerName,
                                                  MQeFields filter,
                                                  long confirmID) throws Exception
```

説明 キュー・マネージャーにより保留になっているメッセージを取得します。このメソッドは、複数のキュー・マネージャーのメッセージを保管できるキュー (たとえば、ストア・アンド・フォワード・キュー) によってのみ呼び出されます。

パラメーター

queueManagerName
キュー・マネージャーの名前の入ったストリング。

filter メッセージ・フィルターを含む MQeFields オブジェクト。

confirmID 確認 ID として使用される long 値。

戻り値 フィルターと一致する保留メッセージを含む MQeMessageObject。

MQeAbstractMessageStore open

構文

```
protected abstract void open(String queueManagerName,
                             String queueName,
                             MQeAttribute attribute) throws Exception
```

説明 メッセージ・ストアをオープンします。このメソッドは、メッセージ・ストアの各ライフ・サイクルの初めに一度だけ呼び出されます。

パラメーター

queueManagerName
親キューを所有するキュー・マネージャーの名前の入ったストリング。

queueName 親キューの名前の入ったストリング

attribute ディスク記憶装置で使用される暗号化および暗号化解除の属性を含む MQeAdminQueueAdminMsg オブジェクト。

戻り値 なし

MQeAbstractMessageStore putMessage

構文

```
protected abstract MQeMsgObject putMessage(MQeMsgObject msgObj,
                                           long confirmID) throws Exception;
```

説明 このメソッドは、指定されたメッセージをメッセージ・ストアに入れます。

保証されたメッセージ送達は、*confirmID* パラメーターの値に依存します。非ゼロ値を渡すと通常のメッセージが受諾されますが、そのメッセージは、後続の確認が受け取られるまでターゲットのメッセージ・ストアにロックされています。ゼロの値を渡すと、後続の確認も必要なくメッセージは送信されますが、メッセージ送達は保証されません。 *confirmID* は、このコマンドの実行時にエラーが生じた場合にも使用されます。書き込み操作で使用した *confirmID* を **undo()** メソッドに渡せば、未確認のメッセージはメッセージ・ストアから除去されます。書き込み操作のたびに、固有の値を使用するようお勧めします。固有の値は、**MQe.uniqueValue()** メソッドを使用して生成できます。メッセージは、メッセージ・レベルのセキュリティーによって保護できます (MQSeries Everyplace のセキュリティーについては、「MQSeries Everyplace

for Multiplatforms プログラミング・ガイド」を参照してください)。セキュリティーは、MQeAttribute オブジェクト、またはこの下位オブジェクトの 1 つを提供することによって定義されます。この属性は、メッセージ書き込み要求の前にメッセージに付加することができます。または、属性パラメーターを使用して、メッセージ・レベルのセキュリティーを使用するように指定できます。属性パラメーターがヌルでない場合、この値はメッセージ書き込み要求の前にメッセージに付加された属性をオーバーライドします。属性パラメーターがヌルの場合、メッセージの送信に対する影響はありません。

パラメーター

msgObject メッセージを含む MQeMsgObject。

confirmID 保証されたメッセージ送達を使用するかどうかを指示する long 値。非ゼロ値であれば、メッセージはターゲットのメッセージ・ストアでロックされ、後続の確認フローまで見えなくなります。ゼロの値であれば、後続の確認も必要なく、メッセージは保管されます。

戻り値 なし

例外

MQeException Except_Duplicate

MQeAbstractmessageStore resetLockedMsgs

構文

```
protected abstract void resetLockedMsgs() throws Exception
```

説明 すべてのロックされたメッセージをリセットします。まれですが、管理メッセージがメッセージ・ストアで、ロック状態になったままの場合があります。たとえば、メッセージの処理中に、キュー・マネージャーまたは JVM が停止することがあります。このメソッドは、メッセージをリセットして、もう一度処理されるようにします。

パラメーター

なし

戻り値 なし

例外 なし

MQeAbstractMessageStore transmitAll

構文

```
protected abstract void transmitAll() throws Exception
```

説明 以下のいずれかの結果、保留メッセージのどれか、またはすべてを送信します。

1. 未確認および再確認としてマークされたメッセージをスキャンする
2. 送信済みとしてマークされたメッセージをスキャンし、再送信する
3. 保留メッセージをスキャンし、送信する

このクラスは、この送信を援助する他のメソッドを 2 つ提供します。

- **transmitMessage()**
- **transmitConfirmMessage()**

パラメーター

なし

戻り値 なし

MQeAbstractmessageStore transmitConfirmMessage

構文

```
protected void transmitConfirmMessage( MQeMsgObject msg ) throws Exception
```

説明 確認付きでメッセージを送信します。

パラメーター

msg 送信されるメッセージを含む MQeMsgObject。

戻り値 なし

MQeAbstractmessageStore transmitMessage

構文

```
protected void transmitMessage( MQeMsgObject msg ) throws Exception
```

説明 確認なしでメッセージを送信します。

パラメーター

msg 送信されるメッセージを含む MQeMsgObject。

戻り値

MQeAbstractmessageStore undo

構文

```
protected abstract void undo(long confirmID) throws Exception
```

説明 このメソッドは、**put()**、**get()**、または **browseAndLock()** コマンドの実行時にエラーが発生した場合に使用することを意図しています。エラーにより、メッセージがターゲットのメッセージ・ストアで、未確認またはロック状態のままになることがあります。このメソッドは、メッセージを失敗した操作の前の状態 (ロック、またはアンロックのいずれでも) にリセットします。未確認の **put()** 操作の場合は、メッセージを削除します。メッセージをリセットするに

は、失敗した操作で使用された *confirmID* を提供しなければなりません。*confirmID* は、各メッセージごとに固有にするようお勧めします。固有の値は、 **MQe.uniqueValue()** メソッドを使用して生成できます。

パラメーター

confirmID 失敗した操作で使用された *confirmID* と同じ long 値。

戻り値 なし

例外

MQeException Except_NotAllowed

MQeAbstractMessageStore updateRetryCount

構文

```
protected abstract void updateRetryCount(MQeFields filter,
                                         MQeAttribute attribute,
                                         int retryCount) throws Exception
```

説明 このメソッドは、管理コードによって内部的に使用され、ロックされたメッセージの再試行数を更新します。メッセージの到着時刻と管理キュー・スレッドが制御を取得する時刻との間に、メッセージが表示されたりブラウズされたり (ロック付きで) する場合に発生することがあります。管理キュー・スレッドはメッセージにアクセスできないため、管理要求を処理できません。これは、失敗と、許可された一回の再試行を行ったことを示します。再試行数は、これを反映して増やされなければなりません。メッセージがロックされて使用できないため、再試行はメッセージ・ストアによって実行されなければなりません。

パラメーター

filter メッセージ・フィルターを含む MQeFields オブジェクト。フィルターには、メッセージの *UID* が含まれていなければなりません。

attribute ヌル、または強制的にオブジェクトに入れるメッセージ・レベルの属性を含む MQeAdminQueueAdminMsg オブジェクト。

retryCount メッセージに入れる新しい再試行数を表す値。

戻り値 なし

例外

MQeException Except_Q_NoMatchingMsg

MQeAbstractMessageStore wrapMsg

構文

```
protected MQeMsgObject wrapMsg(MQeMsgObject msg) throws Exception
```

説明 メッセージをラップしてエンコードします。メッセージがエンコード属性を持つ場合、送信および検索などの操作で、エンコードされ、他のメッセージにラップされる必要があります。サブクラスが、このメソッドをオーバーライドして、索引作成情報をラッピングしているメッセージに追加しようとすることがあります。

パラメーター

msg ラップされるメッセージを含む MQeMsgObject。

戻り値 ラップされたメッセージを含む MQeMsgObject、またはラッパー。

MQeAbstractMessageStore unlockMessage

構文

```
protected abstract MQeMsgObject unlockMessage(MQeFields filter,
          boolean returnMessage) throws Exception
```

説明 このメソッドは、以前にロックされたメッセージをアンロックします。メッセージは、再びすべてのアプリケーションから見えるようになります。一度にアンロックできるメッセージは 1 つだけで、メッセージの *UID* と *lockID* の両方が必要です。メッセージが使用可能でない場合、例外が throw されます。このメソッドは通常、**browseMessagesAndLock()** メソッドと組み合わせて使用します。

パラメーター

filter メッセージ・フィルターを含む MQeFields オブジェクト。操作が成功するためには、このフィルターに *uniqueID* および *lockID* の両方が含まれていなければなりません。

returnMessage ブール値。true の場合、呼び出し側はアンロックされたメッセージが戻されることを希望します。 false の場合は、メッセージは戻される必要はありません。メッセージの取得が、単純なアンロックよりコストのかかるメッセージ・ストアでは、これを使用してパフォーマンスを向上させることができます。

戻り値 アンロックされたメッセージを含む MQeMsgObject。

例外

MQeException Except_Q_NoMatchingMsg

Except_NotAllowed

MQeAbstractMessageStore getStringArgument

構文

```
protected final String getStringArgument()
```

説明 メッセージ・ストアによって使用されるために所有するキューに渡されるパラメーターを取得します。

パラメーター
なし

戻り値 キューの保管ロケーションの入ったストリング。

例外 なし

MQeAbstractMessageStore rule

構文

```
public final MQeQueueRule rule()
```

説明 メッセージ・ストアを所有するキューのルール・オブジェクトを取得します。

パラメーター
なし

戻り値 親キューの動作の一部を決定するために使用されるルールを含むMQeQueueRule オブジェクト。

例外 なし

MQeAbstractMessageStore trace

構文

```
protected final void trace(int key, String str1)
protected final void trace(int key, String str1, String str2)
```

説明 単純化されたトレース

パラメーター

key

str1

str2

戻り値 なし

例外 なし

MQeAbstractMessageStore unWrapMsg

構文

```
protected final MQeMsgObject unWrapMsg(MQeMsgObject wrapper,
                                       MQeAttribute attribute) throws Exception
```

説明 エンコードされたメッセージをアンラップしてデコードします。このメッセージ・ストアを離れるメッセージは、アンラップされる必要があります。

パラメーター

wrapper

attribute

戻り値 アンラップされたメッセージを含む MQeMsgObject。

MQeAbstractMessageStore WrapMsg

構文

```
protected static MQeMsgObject WrapMsg(MQeMsgObject msg) throws Exception
```

説明 メッセージをラップしてエンコードします。メッセージがエンコード属性を持つ場合、送信および検索などの操作で、エンコードされ、他のメッセージにラップされる必要があります。サブクラスが、このメソッドをオーバーライドして、索引作成情報をラッピングしているメッセージに追加しようとすることがあります。

パラメーター

msg ラップされるメッセージを含む MQeMsgObject。

戻り値 ラップされたメッセージを含む MQeMsgObject、またはラッパー。

MQeAdapter

これは、すべての MQSeries Everyplace アダプターが提供する必要のあるメソッドの定義です。新しいアダプターは、MQeAdapter から継承します。

パッケージ **com.ibm.mqe**

メソッド

メソッド	目的
activate	ロードされたアダプターをアクティブにします。
close	アダプターを終了するために使用します。
control	アダプター固有の制御機能を実行します。
checkOption	ファイル・アダプター内でオプションを検査するために使用します。
equals	アダプター・インスタンスで等価性を検査します。
erase	アダプターによってファイルを消去します。
open	アダプターをオープンします。
qualityOfService	qualityOfService オブジェクト (MQeFields オブジェクト) を戻します。
read	アダプターからデータを読み取ります。
readEOF	ファイルを完全に読み取るか、アダプターから EOF 例外が発生するまで読み取ります。
readln	アダプターから改行文字までのデータを読み取ります。
readObject	アダプターからデータを読み取り、オブジェクトを戻します。
status	アダプター状況情報を要求します。
write	アダプターによってデータを書き込みます。
writeln	アダプターにデータと改行文字を書き込みます。
WriteObject	オブジェクトからのデータをアダプターに書き込みます。

MQeAdapter activate

構文

```
public void activate( String fileId,
                      Object parameter,
                      Object option,
                      int  value1,
                      int  value2 ) throws Exception
```

説明 これは、アダプターの活動化を指示するために使用します。

注: このエントリー・ポイントは、 MQSeries Everyplace オブジェクト・ライブラリーによって使用されますが、アプリケーション・プログラムによっては使用されません。

パラメーター

fileId ファイルの ID です。

parameter アダプターのパラメーター、またはヌルです。

options アダプターのオプション、またはヌルです。

value1 整数値、または設定しないことを示す -1 です。

value2 整数値、または設定しないことを示す -1 です。

戻り値 なし

例外

IOException デバイスが作動不能か入出力エラーが発生しています。

MQeAdapter checkOption

構文

```
1. protected boolean checkOption( String what ) throws Exception
2. protected boolean checkOption( Object options,
                                  String what ) throws Exception
```

説明 この保護されたメソッドは、新しい MQSeries Everyplace アダプターを書き込むときに使用されます。メソッドはマッチング・オプションを調べ、見つかれば true を戻します。メソッドには 2 つの形式があります。

1. **activate()** メソッドに指定されたオプションを調べるもの
2. *options* パラメーターのオプションを調べるもの

注: このエントリー・ポイントは、MQeAdapter の子孫によって使用され、アプリケーション・プログラムによっては使用されません。

パラメーター

option この操作のオプションです。

what 検査されるオプションの入ったストリングです。

戻り値 ブール型の戻りコード:

true オプションが検出されました。

false オプションは検出されませんでした。

例外

IOException ファイルのクローズ中にエラーが発生しました。

MQeAdapter close

構文

```
public void close( Object options ) throws Exception
```

説明 ファイルをアンバインドします。

MQeAdapter 基本クラスは、″not supported″ 例外を出します。新しいアダプターは、適切であればこのメソッドを上書きします。

注: このエントリー・ポイントは、 MQSeries Everyplace オブジェクト・ライブラリーによって使用されますが、アプリケーション・プログラムによっては使用されません。

パラメーター

options アダプターのオプション、またはヌルです。

戻り値 なし

例外

IOException デバイスが作動不能か入出力エラーが発生しています。

MQeAdapter control

構文

```
public Object control(Object options,
                      Object ctrlObj ) throws Exception
```

説明 MQeAdapter 基本クラスは、″not supported″ 例外を出します。新しいアダプターは、適切であればこのメソッドを上書きします。

注: このエントリー・ポイントは、 MQSeries Everyplace オブジェクト・ライブラリーによって使用されますが、アプリケーション・プログラムによっては使用されません。

パラメーター

options アダプターのオプション、またはヌルです。

ctrlObj 制御機能 (各アダプター・タイプに固有) のためにアダプターによって使用されるオブジェクトです。

戻り値 アダプター・タイプに従属するオブジェクトまたはヌルです。

例外

IOException デバイスが作動不能か入出力エラーが発生しています。

MQeAdapter equals

構文

```
public boolean equals( Object item )
```

説明 このメソッドは、このアダプターとの等価性検査を実行するために使用されます。

item がストリングの場合、MQeAdapter 基本クラスは、 *fileID* を提供された *item* と比較します。そうでない場合は、base object equals メソッドが呼び出されます。

注: このエントリー・ポイントは、 MQSeries Everyplace オブジェクト・ライブラリーによって使用されますが、アプリケーション・プログラムによっては使用されません。

パラメーター

item 比較されるオブジェクトです。

戻り値 ブール値の true または false です。

例外

IOException デバイスが作動不能か入出力エラーが発生しています。

MQeAdapter erase

構文

```
public void erase( Object options ) throws Exception
```

説明 このメソッドは既存のファイルを削除するために使用されます。

MQeAdapter 基本クラスは ″not supported″ 例外を出します。新しいアダプターは、適切であればこのメソッドを上書きします。

注: このエントリー・ポイントは、 MQSeries Everyplace オブジェクト・ライブラリーによって使用されますが、アプリケーション・プログラムによっては使用されません。

パラメーター

options アダプターのオプション、またはヌルです。

戻り値 なし

例外

IOException デバイスが作動不能か入出力エラーが発生しています。

MQeAdapter open

構文

```
public void open( Object options ) throws Exception
```

説明

このメソッドは、アダプターによってファイルにバインドするために使用します。

MQeAdapter 基本クラスは、″not supported″ 例外を出します。新しいアダプターは、適切であればこのメソッドを上書きします。

注: このエントリー・ポイントは、 MQSeries Everyplace オブジェクト・ライブラリーによって使用されますが、アプリケーション・プログラムによっては使用されません。

パラメーター

options アダプターのオプション、またはヌルです。

戻り値 なし

例外

IOException デバイスが作動不能か入出力エラーが発生しています。

MQeAdapter qualityOfService

構文

```
public void qualityOfService( Object options ) throws Exception
```

説明

このメソッドは、アダプターのインスタンスに関連したサービス品質オブジェクトを取得するために使用します。

注: このエントリー・ポイントは、 MQSeries Everyplace オブジェクト・ライブラリーによって使用されますが、アプリケーション・プログラムによっては使用されません。

パラメーター

options アダプターのオプション、またはヌルです。

戻り値 サービス品質オブジェクトです。

例外 なし

MQeAdapter read

構文

```
public byte[] read( Object options,
                                int value0 ) throws Exception
```

説明

このメソッドは指定のファイルからレコードを読み取るために使用します。

MQeAdapter 基本クラスは、″not supported″ 例外を出します。新しいアダプターは、適切であればこのメソッドを上書きします。

注: このエントリー・ポイントは、 MQSeries Everyplace オブジェクト・ライブラリーによって使用されますが、アプリケーション・プログラムによっては使用されません。

パラメーター

options アダプターのオプション、またはヌルです。

value0 書き込まれるレコード番号または -1 です。

戻り値 ファイル・オブジェクトから読み取られるデータ・バイトの入ったサービス品質バイト配列。

例外

IOException デバイスが作動不能か入出力エラーが発生しています。

EOFException このファイルの終わりが過ぎています。

MQeAdapter readEOF

構文

```
public byte[] readEOF( Object options ) throws Exception
```

説明

このメソッドは、ファイルを EOF 条件に達するまで読み取るために使用します。

MQeAdapter 基本クラスは、″not supported″ 例外を出します。新しいアダプターは、適切であればこのメソッドを上書きします。

注: このエントリー・ポイントは、 MQSeries Everyplace オブジェクト・ライブラリーによって使用されますが、アプリケーション・プログラムによっては使用されません。

パラメーター

options アダプターのオプション、またはヌルです。

戻り値 ファイル・データ・バイトの入ったバイト配列です。

例外

IOException デバイスが作動不能か入出力エラーが発生しています。

MQeAdapter readln

構文

```
public String readln( Object options ) throws Exception
```

説明

このメソッドは指定のファイルからレコードを読み取るために使用します。

MQeAdapter 基本クラスは、″not supported″ 例外を出します。新しいアダプターは、適切であればこのメソッドを上書きします。

注: このエントリー・ポイントは、 MQSeries Everyplace オブジェクト・ライブラリーによって使用されますが、アプリケーション・プログラムによっては使用されません。

パラメーター

options アダプターのオプション、またはヌルです。

戻り値 ファイルから読み取られるデータ・バイトの入ったストリングです。

例外

IOException デバイスが作動不能か入出力エラーが発生しています。

EOFException このファイルの終わりが過ぎています。

MQeAdapter readObject

構文

```
public Object readObject( Object options ) throws Exception
```

説明

このメソッドは、指定のファイルからオブジェクトを読み取るために使用します。

MQeAdapter 基本クラスは、″not supported″ 例外を出します。新しいアダプターは、適切であればこのメソッドを上書きします。

注: このエントリー・ポイントは、 MQSeries Everyplace オブジェクト・ライブラリーによって使用されますが、アプリケーション・プログラムによっては使用されません。

パラメーター

options アダプターのオプション、またはヌルです。

戻り値 ファイルから読み取られるデータの入ったオブジェクトです。

例外

IOException デバイスが作動不能か入出力エラーが発生しています。

EOFException このファイルの終わりが過ぎています。

MQeAdapter status

構文

```
public Object status( Object options ) throws Exception
```

説明

このメソッドは、アダプター状況情報をストリングとして戻すために使用します。

MQeAdapter 基本クラスは、″not supported″ 例外を出します。新しいアダプターは、適切であればこのメソッドを上書きします。

注: このエントリー・ポイントは、 MQSeries Everyplace オブジェクト・ライブラリーによって使用されますが、アプリケーション・プログラムによっては使用されません。

パラメーター

options アダプターのオプション、またはヌルです。すべてのアダプターは必ず以下のオプションをサポートする必要があります。

MQe_File_NETWORK
ヌルまたはネットワーク・タイプ (たとえば、TCPIP) を戻します。

MQe_File_BYTECOUNTS
アダプターによって読み取られるおよび (または) 書き込まれるバイト数を戻します。

戻り値 ファイルから読み取られるデータ・バイトの入ったストリングです。

例外

IOException デバイスが作動不能か入出力エラーが発生しています。

EOFException このファイルの終わりが過ぎています。

MQeAdapter write

構文

```
public void write( Object options,
                   int value0,
                   byte data[] ) throws Exception
```

説明

このメソッドは、指定のファイルにデータを書き込むために使用します。

MQeAdapter 基本クラスは、″not supported″ 例外を出します。新しいアダプターは、適切であればこのメソッドを上書きします。

注: このエントリー・ポイントは、 MQSeries Everyplace オブジェクト・ライブラリーによって使用されますが、アプリケーション・プログラムによっては使用されません。

パラメーター

options アダプターのオプション、またはヌルです。

value0 書き込まれるレコード番号または -1 です。

data 書き込まれるデータの入ったバイト配列です。

戻り値 なし

例外

IOException デバイスが作動不能か入出力エラーが発生しています。

EOFException このファイルの終わりが過ぎています。

MQeAdapter writeln

構文

```
public void Writeln( Object options,
                     String data ) throws Exception
```

説明

このメソッドは、指定のファイルにデータを書き込むために使用します。

MQeAdapter 基本クラスは、″not supported″ 例外を出します。新しいアダプターは、適切であればこのメソッドを上書きします。

注: このエントリー・ポイントは、 MQSeries Everyplace オブジェクト・ライブラリーによって使用されますが、アプリケーション・プログラムによっては使用されません。

パラメーター

options アダプターのオプション、またはヌルです。

data 書き込まれるデータの入ったストリングです。

戻り値 なし

例外

IOException デバイスが作動不能か入出力エラーが発生しています。

EOFException このファイルの終わりが過ぎています。

MQeAdapter writeObject

構文

```
public void writeObject( Object options,
                         Object data ) throws Exception
```

説明

このメソッドは、指定のファイルにオブジェクトを書き込むために使用します。

MQeAdapter 基本クラスは、″not supported″ 例外を出します。新しいアダプターは、適切であればこのメソッドを上書きします。

注: このエントリー・ポイントは、 MQSeries Everyplace オブジェクト・ライブラリーによって使用されますが、アプリケーション・プログラムによっては使用されません。

パラメーター

options アダプターのオプション、またはヌルです。

data 書き込まれるデータの入ったオブジェクト。

戻り値 なし

例外

IOException デバイスが作動不能か入出力エラーが発生しています。

EOFException このファイルの終わりが過ぎています。

MQeAdminMsg

このクラスは、基本的な MQeAdminMsg を作成するために使用します。MQeMsgObject を拡張し、管理メッセージの基本を提供します。このクラスから、異なるタイプのリソースを管理するためのものが作成されます。

パッケージ **com.ibm.mqe**

このクラスは、 MQeMsgObject の下位クラスです。

- 定数と変数
- コンストラクター
- メソッド

定数と変数

MQeAdminMsg には、MQeMsgObject によって提供され継承される定数と変数だけでなく、以下の定数と変数が備えられています。

メッセージ内の追加のフィールド

Action:

実行される管理アクション (int)

```
public final static  String  Admin_Action;
```

Errors: アクションからのエラー結果 (MQeFields)

```
public final static  String  Admin_Errors;
```

MaxAttempts:

要求が試行される最大回数 (int)

```
public final static  String  Admin_MaxAttempts;
```

Parms:

アクションの入出力パラメーター (MQeFields)。アクションで必要とされる、またはアクションの結果として戻される管理対象リソースの特性が入っています。

```
public final static  String  Admin_Parms;
```

RC: アクション・コードの結果 (バイト)

```
public final static  String  Admin_RC;
```

Reason:

失敗の理由 (Unicode)

```
public final static  String  Admin_Reason;
```

TargetQMgr:

アクションを実行するときの対象となるキューの名前 (ASCII)。

```
public final static  String  Admin_TargetQMgr
```

管理アクションの基本タイプ

Create:
リソースを作成します。

```
public final static  int Action_Create;
```

Delete: リソースを削除します。

```
public final static  int Action_Delete;
```

Inquire:
要求されたリソースの特性を戻します。

```
public final static  int Action_Inquire;
```

InquireAll:
リソースのすべての特性を戻します。

```
public final static  int Action_InquireAll;
```

Update:
リソースの特性を更新します。

```
public final static  int Action_Update;
```

管理対象リソースのフィールド名 (ASCII)

Name:
管理対象リソース名。これは、管理対象リソースの特性であり、フィールドは *Admin_Parms* フィールド内になければなりません。

```
public final static  String Admin_Name;
```

管理対象リソースの Java クラス (ASCII)

Class:
管理対象リソース・クラス。これは、管理対象リソースの特性であり、フィールドは *Admin_Parms* フィールド内になければなりません。

```
public final static  String Admin_Class;
```

戻りコード

Fail: アクションは失敗しました。*Reason* を参照してください。

```
public final static  int RC_Fail;
```

Mixed: アクションは一部成功しました。 *Reason* を参照してください。

```
public final static  int RC_Mixed;
```

Success:

アクションは成功しました。

```
public final static  int RC_Success;
```

コンストラクター

MQeAdminMsg

構文

```
public MQeAdminMsg() throws Exception
```

説明 コンストラクターは、デフォルトの MQeAdminMsg を作成して初期設定します。

パラメーター

なし

戻り値 なし

例外

java.lang.Exception さまざまなものがあります。

例

```
class MyApplication
{
  MQeAdminMsg aMsg = new MQeAdminMsg();
}
```

メソッド

メソッド	目的
characteristics	リソースの特性の入った MQeFields オブジェクトを戻します。
create	**Admin_Create** アクションを実行する管理メッセージを設定します。
delete	**Admin_Delete** アクションを実行する管理メッセージを設定します。
duplicate	*replyType* パラメーターによって指定されたタイプの新しいメッセージを作成します。
getAction	実行される、または実行された管理アクションを戻します。
getErrorFields	エラー・フィールド・オブジェクトへの参照を戻します。
getFieldInError	フィールドを処理するときにエラーが発生したフィールド名を戻します。
getInputFields	入力フィールド・オブジェクトへの参照を戻します。
getName	管理対象リソースの名前を取得します。
getOutputFields	出力フィールド・オブジェクトへの参照を戻します。
getRC	アクションの結果の戻りコードを戻します。

メソッド	目的
getReason	エラーが発生したときにそのエラーの理由を戻します。
getTargetQMgr	要求を処理するキュー・マネージャーを戻します。
inquire	**Action_Inquire** アクションを実行する管理メッセージを設定します。
inquireAll	**Action_InquireAll** アクションを実行する管理メッセージを設定します。
setAction	管理アクションを設定して実行します。
setName	アクションを実行するときの対象となるリソースの名前。
setTargetQMgr	要求を処理するキュー・マネージャーを戻します。
update	**Action_Update** アクションを実行する管理メッセージを設定します。

MQeAdminMsg characteristics

構文

```
public MQeFields characteristics() throws Exception
```

説明 リソースの特性の入った MQeFields オブジェクトを戻します。リソースのフィールド名とタイプの完全なセットは、結果の MQeFields オブジェクトから判別されます。(それには、各特性の値は含まれていません)。

パラメーター

なし

戻り値 リソースの有効な特性。

例外

java.lang.Exception	さまざまなものがあります。

例

```
class MyApplication
{

  MQeFields  chars  = msg.characteristics();
  Enumeration fields    = chars.fields()
    while ( fields.hasMoreElements() )
  {
    System.out.println( "Contains field: "+
              (String)fields.nextElement() );
  }
}
```

MQeAdminMsg create

構文

```
public void create( MQeFields parms ) throws Exception
```

説明 **Admin_Create** アクションを実行する管理メッセージを設定します。*parms* パラメーターに指定した特性の、新しい管理対象リソースの作成を試行します。

パラメーター

parms 管理対象リソースのデフォルト設定とは異なる設定を必要とする特性の名前と値の組みの入った MQeFields オブジェクト。リソースの名前は *parms* に含めることができますが、**setName** メソッドによって設定することもできます。

戻り値 なし

例外

java.lang.Exception さまざまなものがあります。

例

```
class MyApplication
{
  ...
  // Create ExampleQ
  MQeFields parms = new MQeFields();
  msg.setName( "ExampleQM", "ExampleQ" );
  parms.putUnicode(  MQeQueueAdminMsg.Queue_Description,
  "a new description ..." );
  // Set the action required and its parameters
  // into the message
  msg.create( parms );
}
```

MQeAdminMsg delete

構文

```
public void delete( MQeFields parms ) throws Exception
```

説明 **Admin_Delete** アクションを実行する管理メッセージを設定します。管理対象リソースの削除を試行します。

パラメーター

parms MQeFields オブジェクト。 setName メソッドによって設定されていない場合、削除する管理対象リソースの名前を含める必要があります。

戻り値 なし

例外

java.lang.Exception さまざまなものがあります。

例

```
class MyApplication
{
  ...
```

```
  // Delete ExampleQ
  MQeFields parms = new MQeFields();
  msg.setName( "ExampleQM", "ExampleQ" );
  msg.delete( parms );
}
```

MQeAdminMsg duplicate

構文

```
public MQeFields duplicate( String replyType ) throws Exception
```

説明

replyType パラメーターによって指定されたタイプの新しいメッセージを作成します。ヌルの場合、このメッセージと同じタイプのメッセージが戻されます。フィールドは、*UID* を除き、すべて複写されます。

注: MQeFields.copy メソッドを使用して、メッセージのシャロー・コピーのみが作成されます。

パラメーター

replyType 戻されるメッセージのタイプ、またはこのメッセージと同じ場合はヌル。

戻り値 複製されたメッセージ

例外 ClassNotFoundException

例

```
class MyApplication
{
  // Create a message as the same type as this one
  MQeQueueAdminMsg reply =
        (MQeQueueAdminMsg).requestMsg.duplicate( null );
}
```

MQeAdminMsg getAction

構文

```
public int getAction( )
```

説明 実行される、または実行された管理アクションを戻します。

パラメーター

なし

戻り値 MQeAdminMsg からの *Admin_Action* フィールド、または設定されていない場合は Action_Unknown。

例外 なし

例

```
class MyApplication
 {
   ...
   int action = requestMsg.getAction();
    switch ( action )
    {
      case Create :
        performCreate();
        break;
      case Delete :
        performDelete();
        ...
    }
  }
```

MQeAdminMsg getErrorFields

構文

```
public MQeFields getErrorFields()
```

説明 エラー・フィールド・オブジェクトへの参照を戻します。

エラー・フィールドには、アクションの処理のときに発生した副次的な問題に関連したエラーもすべて含まれます。たとえば、2 つの特性を変更する要求を出したときに、1 つの要求は成功し、もう 1 つが失敗する場合、 *ErrorFields* には失敗した要求の詳細が示されます。エラーのあるフィールドの名前は、*Admin_Parms* フィールドと一致します。

getRC メソッドを使用して、アクションの結果全体を調べてください。

パラメーター

なし

戻り値 空の MQeFields オブジェクト、または MQeAdminMsg からの *Admin_Errors* フィールド。

例外 なし

例

```
class MyApplication
{
  if ( replyMsg.getRC() != 0 )
  {
    MQeFields  errs = replyMsg.getErrorFields();
    Enumeration fields = errs.fields()
    while ( fields.hasMoreElements() )
    {
      String errF = (String)fields.nextElement()
      System.out.println( "Field: "+
                  errF+
                  "failed with error "+
```

```
                    fields.getAscii( Msg_RC ) );
    }
  }
}
```

MQeAdminMsg getFieldInError

構文

```
public String[] getFieldInError( String fieldName )
```

説明 このメソッドは、**getRC** に対して RC_Fail または RC_Mixed が戻された後、個々のエラーに関する情報を得るために使用します。フィールドを処理するときにエラーが発生したフィールド名を戻します。処理されたフィールドが配列されると、同じ数のエレメントが入った、対応するストリング配列が戻されます。処理されたフィールドが配列されない場合、戻される配列には 1 つのエレメントしか含まれません。フィールドにエラーがない場合、 ヌルが戻されます。

パラメーター

fieldname エラーをテストするフィールドの名前

戻り値 指定されたフィールドを処理するときに発生したエラーの入ったストリング配列。

例外 なし

例

```
class MyApplication
{
  if ( replyMsg.getRC() != 0 )
  {
    String  fieldName = MQeQueueAdminMsg.Queue_Priority
    String[] errs = replyMsg.getFieldInError( fieldName );
    if ( errs != null )
         System.out.println( "Error setting priority"+ errs[0]
  }
}
```

MQeAdminMsg getInputFields

構文

```
public MQeFields getInputFields()
```

説明 入力フィールド・オブジェクトへの参照を戻します。入力フィールド・オブジェクトには、アクションに必要な入力パラメーターが入っています。

パラメーター

なし

戻り値 アクションに必要な入力パラメーターの入った MQeFields オブジェクトへの参照。

例外 なし

例

```
class MyApplication
{
  MQeFields parms = requestMsg.getInputFields()
}
```

MQeAdminMsg getMaxAttempts

構文

```
public int getMaxAttempts( )
```

説明 要求時にリソースが使用できないために保留される要求の、再試行される最大回数を取得します。

パラメーター

なし

戻り値 MQeAdminMsg からの *Admin_MaxAttempts* フィールドが戻されるか、または設定されていない場合はデフォルトの 1 が戻されます。

例外 なし

例

```
class MyApplication
{
  ...
  int tries = requestMsg.getMaxAttempts();
  ...
}
```

MQeAdminMsg getName

構文

```
public String getName( )
```

説明 管理対象リソースの名前を取得します。設定されていない場合ヌルです。

パラメーター

なし

戻り値 MQeAdminMsg からの *Admin_Name* フィールド、または設定されていない場合はヌル。

例外 なし

例

```
class MyApplication
{
  ...
  String name = requestMsg.getName();
  ...
}
```

MQeAdminMsg getOutputFields

構文

```
public MQeFields getOutputFields()
```

説明 出力フィールド・オブジェクトへの参照を戻します。 OutputFields には、要求の入力パラメーターと、要求の結果の両方が入れられます。

パラメーター

なし

戻り値 アクションの結果

例外 なし

例

```
class MyApplication
{
  MQeFields parms = replyMsg.getOutputFields()
  if (parms.contains( MQeQueueAdminMsg.desc ) )
  {
    System.out.println("Queue description: "+
       parms.getUnicode(MqeQueueAdminMsg.Desc) );
  }
}
```

MQeAdminMsg getRC

構文

```
public int getRC( ) throws Exception
```

説明 アクションの結果のコードを戻します。

パラメーター

なし

戻り値 戻りコード。

取り得る値は以下のとおりです。

```
  public final static  int RC_Success;
  public final static  int RC_Fail;
  public final static  int RC_Mixed;
```

例外

java.lang.Exception	さまざまなものがあります。

例

```
class MyApplication
{
  ...
  int rc = ReplyMsg.getRC();

  if (rc != ReplyMsg.RC_success)
    String error = replyMsg.getReason();
  ....
}
```

MQeAdminMsg getReason

構文

```
public String getReason( )
```

説明 エラーが発生したときにそのエラーの理由を戻します。

パラメーター

なし

戻り値 ストリング。一般的にはエラーを生じた例外です。例外が MQeException タイプの場合、ストリングの開始位置 ″Code=nnn;″ に MQeException コードが組み込まれます。

例外 なし

例

```
class MyApplication
{
  ...
  int rc = replyMsg.getRC();

  if (rc != replyMsg.RC_success)
    String error = replyMsg.getReason();
  ...
}
```

MQeAdminMsg getTargetQMgr

構文

```
public String getTargetQMgr( ) throws MQeException
```

説明 要求を処理するキュー・マネージャーを戻します。

パラメーター

なし

戻り値 要求を処理するキュー・マネージャー。

例外

MQeException	Except_Type, "wrong field type"

```
                                        Except_NotFound, Item + " not found"
```

例

```
class MyApplication
{
  try
  {
    String targetQMgr = requestMsg.getTargetQMgr();
  }
  catch ( MQeException e)
  {
    System.out.prinln("Target queue manager not set")
  }
}
```

MQeAdminMsg inquire

構文

```
public void inquire( MQeFields parms ) throws Exception
```

説明 **Action_Inquire** アクションを実行する管理メッセージを設定します。

パラメーター

parms 照会される管理対象リソースの特性の名前。 **setName()** メソッドによって管理対象リソースの名前が設定されていない場合は、その名前をパラメーターに組み込むこともできます。

戻り値 なし

例外 NullPointerException

例

```
class MyApplication
{
  ...
  // Request the value of description and max queue depth
  MQeFields parms = new MQeFields();
  parms.putUnicode(  MQeQueueAdminMsg.Queue_Description,  null );
  parms.putInt(  MQeQueueAdminMsg.Queue_MaxQDepth, 0 );

  // set the name of the queue to inquire on
  msg.setName( "ExampleQM", "ExampleQ" );

  // Set the action required and its parameters
  // into the message
  msg.inquire( parms );
}
```

MQeAdminMsg inquireAll

構文

```
public void inquireAll( MQeFields parms ) throws Exception
```

説明 **Action_InquireAll** アクションを実行する管理メッセージを設定します。InquireAll アクションは、管理対象リソースのすべての特性を戻します。

パラメーター

parms 照会するリソースの名前を含めるか、名前がすでに **setName()** メソッドによって設定されている場合はヌルにします。

戻り値 なし

例外 NullPointerException

例

```
class MyApplication
{
  ...
  // set the name of the queue to inquire on
  msg.setName( "ExampleQM", "ExampleQ" );

  // Set the action required and its parameters
  // into the message
  msg.inquireAll( new MQeFields() );
}
```

MQeAdminMsg setAction

構文

```
public void setAction(int action )
```

説明 実行する管理アクションを設定します。 MQeAdminMsg の *Admin_Action* フィールドを設定します。

パラメーター

action 取り得る値は以下のとおりです。

```
  public final static  int Action_Create;
  public final static  int Action_Delete;
  public final static  int Action_Inquire;
  public final static  int Action_InquireAll;
  public final static  int Action_Update;

// additional actions can implemented in subclass
```

戻り値 なし

例外

java.lang.Exception さまざまなものがあります。

例

```
class MyApplication
{
  ...
  MQeAdminMsg requestMsg = new MQeAdminMsg()
  requestMsg.setAction(MQeAdminMsg.Action_Inquire);
  ...
}
```

MQeAdminMsg setName

構文

```
public void setName( String resourceName ) throws Exception
```

説明 アクションを実行するときの対象となるリソースの名前。

パラメーター

resourceName
リソースの名前。

戻り値 なし

例外

java.lang.Exception さまざまなものがあります。

例

```
class MyApplication
{
  ...
  // Delete a queue
  MQeFields parms = new MQeFields();

  // Set the action required and its parameters
  // into the message
  MQeQueueManagerAdminMsg msg = new MQeQueueManagerAdminMsg();
  msg.inquireAll( parms );
  msg.setName( "ExampleQM" );
  ...
}
```

MQeAdminMsg setTargetQMgr

構文

```
public String setTargetQMgr( String targetQMgr ) throws Exception
```

説明 要求を処理するキュー・マネージャーを設定します。

パラメーター

targetQMgr 要求を処理するキュー・マネージャーの名前。

戻り値 なし

例外 なし

例

```
class MyApplication
{
  MQeQueueAdminMsg requestMsg = new MQeQueueAdminMsg();
  requestMsg.setTargetQMgr("ExampleQM");
  requestMsg.setName("ExampleQM", "ExampleQ" );
  requestMsg.create( new MQeFields() );
}
```

MQeAdminMsg update

構文

```
public void update( MQeFields parms ) throws Exception
```

説明 **Action_Update** アクションを実行して、 *parms* にある管理対象リソース特性に基づいて管理対象リソース特性の更新を試行する管理メッセージを設定します。

パラメーター

parms 更新される特性。管理対象リソースの名前が設定されていない場合、パラメーターに組み込むことができます。

戻り値 なし

例外 NullPointerException.

例

```
class MyApplication
{
  ...
  // Setname of resource to be managed
  msg.setName( "ExampleQM", "ExampleQ" );

  // Change the value of description
  MQeFields parms = new MQeFields();
  parms.putUnicode( MQeQueueAdminMsg.Queue_Desc, "Change description ... );

  // Set the action required and its parameters
  // into the message
  msg.update( parms );
}
```

MQeAttribute

このクラスは、属性オブジェクトを作成するときに使います。このオブジェクトには、認証、暗号化、および圧縮を行うメカニズムが入っています。 MQeAttribute オブジェクトは、 チャネル、キュー、メッセージ、および MQeFields オブジェクトに関連づけることができます。

パッケージ **com.ibm.mqe**

このクラスは、MQe の下位クラスです。

コンストラクター

MQeAttribute

構文

```
public MQeAttribute( MQeAuthenticator authenticator,
                     MQeCryptor cryptor,
                     MQeCompressor compressor
                   ) throws Exception
```

説明 MQeAttribute オブジェクトを構成します。

パラメーター

authenticator MQeAuthenticator オブジェクトへのオブジェクト参照

cryptor MQeCryptor オブジェクトへのオブジェクト参照

compressor MQeCompressor オブジェクトへのオブジェクト参照

戻り値 なし

例外

MQeException さまざまな活動化エラー

IOException プロトコル・タイプに応じたさまざまな入出力エラー

例

```
class MySampleClass
 {
 ...
 MQeAttribute attribute = new MQeAttribute( null,
                          new MQeXorCryport( ),
                          new MQeRleCompressor( ) );
 ...
 MQeChannel channel = new MQeChannel( aAttribute,
                      "HTTP://test.server.ibm.com:8080" );
 ...
 }
```

メソッド

メソッド	目的
authenticatedID	認証された ID であるストリングを戻します。
activate	MQeAttribute オブジェクトを活動化にします。
change	この MQeAttribute オブジェクトの特性を変更するために使用します。
close	このオブジェクトによって使用されるリソースを解放します。
decodeData	提供されたデータを復号または圧縮解除 (またはその両方) します。
encodeData	提供されたデータを暗号化または圧縮 (またはその両方) します。
equals	ある MQeAttribute オブジェクトの設定をこの設定と比較します。
getAuthenticator	オーセンティケーターへのオブジェクト参照を取得します。
getCompressor	圧縮機能へのオブジェクト参照を取得します。
getCryptor	暗号化機能へのオブジェクト参照を取得します。
setKey	鍵を属性に関連づけます。

MQeAttribute activate

構文

```
public void activate( MQeRule rule,
                      MQeAuthenticator authenticator,
                      MQeCryptor cryptor,
                      MQeCompressor compressor) throws Exception
```

説明 MQeAttribute オブジェクトを活動化にします。

パラメーター

rule この属性によって使用される MQeRule オブジェクトへのオブジェクト参照

authenticator MQeAuthenticator オブジェクトへのオブジェクト参照

cryptor MQeCryptor オブジェクトへのオブジェクト参照

compressor MQeCompressor オブジェクトへのオブジェクト参照

戻り値 なし

例外

MQeException さまざまな活動化エラー

IOException プロトコル・タイプに応じたさまざまな入出力エラー

例

```
class MySampleClass
  {
  ...
  MQeAttribute attribute = new MQeAttribute( null,
                                             new MQeXorCryport( ),
                                             new MQeRleCompressor( ) );
  ...
  MQeChannel channel = new MQeChannel( attribute,
                                       "HTTP://test.server.ibm.com:8080" );
  ...
  }
```

MQeAttribute authenticatedID

構文

```
public String authenticatedID( )
```

説明 このメソッドは、認証された ID であるストリングを戻すか、認証されていない場合はヌルを戻します。通常これは、データが存在する場合、または特定のユーザーにのみ実行が許可されているプロセスが存在する場合に、チャネルのサーバー・サイドで使用されます。

パラメーター

なし

戻り値 認証された ID であるストリング、またはヌル

例外 なし

MQeAttribute change

構文

```
public synchronized void change( MQeChannel channel,
                                 MQeRule rule,
                                 MQeAttribute attribute) throws Exception
```

説明 このメソッドは、MQeAttribute オブジェクトの特性を変更するために呼び出されます。つまり、MQeAttribute オブジェクトによって使用されるルール、オーセンティケーター、暗号化機能、または圧縮機能を変更する場合です。

channel パラメーター がヌルでない場合、チャネルのリモート・エンドは特性の変更に同意しますが、そうでない場合は例外が発生します。

パラメーター

channel 通信に使用されるチャネルへのオブジェクト参照

rule 変更が許可されていることを確認するために使用される MQeRules オブジェクト

注: 以前の MQeRules オブジェクトは新しい MQeRules オブジェクトを許可します。

attribute MQeAttribute オブジェクトへの参照

戻り値 なし

例外

MQeException Except_Rule, ″Disallowed by rule″

属性によって使用されるオーセンティケーター、暗号化機能、または圧縮機能 (またはこれらすべて) に応じて異なります。

MQeAttribute close

構文

```
public void close( ) throws Exception
```

説明 オーセンティケーターによって使用されるリソースをクローズし解放します。

パラメーター

なし

戻り値 なし

例外

MQeException Invalid または NotAllowed

MQeAttribute decodeData

構文

```
public byte[] decodeData( MQeChannel channel,
                                    byte data[],
                                    int offset,
                                    int count ) throws Exception
```

説明 このメソッドは、*data*、 *offset* および長さ *count* で示されるバイトをデコードする (復号または圧縮解除する (またはその両方)) ときに呼び出されます。

注: このメソッドは内部での使用のためのものであり、通常は、アプリケーションによって呼び出されることはありません。

パラメーター

channel エンコードされたデータまたはヌルを受け取るために使用されるチャネルへのオブジェクト参照

data デコードするデータを含むバイト配列へのオブジェクト参照

offset データ配列での開始バイトを指定する整数索引

count デコードするバイト数を示す整数カウント

戻り値 なし

例外 属性によって使用されるオーセンティケーター、暗号化機能、または圧縮機能 (またはこれらすべて) に応じて異なります。

MQeAttribute encodeData

構文

```
public byte[] encodeData( MQeChannel channel,
                          byte       data[],
                          int        offset,
                          int        count ) throws Exception
```

説明 *data*、*offset*、および長さ *count* で示されるバイトをエンコードする (暗号化または圧縮 (またはその両方)) ときに呼び出されます。

注: このメソッドは内部での使用のためのものであり、通常は、アプリケーションによって呼び出されることはありません。

パラメーター

channel エンコードされたデータまたはヌルを送信するために使用されるチャネルへのオブジェクト参照

data エンコードするデータを含むバイト配列へのオブジェクト参照

offset データ配列での開始バイトを指定する整数索引

count エンコードするバイト数を示す整数カウント

戻り値 なし

例外 属性によって使用されるオーセンティケーター、暗号化機能、または圧縮機能 (またはこれらすべて) に応じて異なります。

MQeAttribute equals

構文

```
public boolean equals( Object thisItem )
```

説明 このメソッドは、*thisItem* とこの MQeAttribute オブジェクトの、等価性を比較するために呼び出されます。

パラメーター

thisItem 通常は MQeAttribute オブジェクトへのオブジェクト参照。

戻り値 ブール値:

true 等しいことを暗黙指定します。

false 等しくないことを暗黙指定します。

例外 属性によって使用されるオーセンティケーター、暗号化機能、または圧縮機能 (またはこれらすべて) に応じて異なります。

MQeAttribute getAuthenticator

構文

```
public MQeAuthenticator getAuthenticator( )
```

説明 これは、この属性によって使用されるオーセンティケーターへのオブジェクト参照を戻すか、オーセンティケーターがない場合はヌルを戻すために呼び出されます。

パラメーター
なし

戻り値 MQeAuthenticator オブジェクト参照、またはヌル。

例外 なし

MQeAttribute getCompressor

構文

```
public MQeCompressor getCompressor( )
```

説明 これは、この属性によって使用される圧縮機能へのオブジェクト参照を戻すか、圧縮機能がない場合はヌルを戻すために呼び出されます。

パラメーター
なし

戻り値 MQeCompressor オブジェクト参照、またはヌル。

例外 なし

MQeAttribute getCryptor

構文

```
public MQeCryptor getCryptor( )
```

説明 これは、この属性によって使用される暗号化機能へのオブジェクト参照を戻すか、暗号化機能がない場合はヌルを戻すために呼び出されます。

パラメーター
なし

戻り値 MQeCryptor オブジェクト参照、またはヌル。

例外 なし

MQeAttribute setKey

構文

```
public void setKey(MQeKey key)
```

説明 このメソッドは、鍵を属性に関連づけます。属性に暗号化機能がある場合に必要です。

パラメーター

key	属性の暗号化機能によって使用される key オブジェクト

戻り値 なし

例外

MQeException	`NotAllowed` (鍵がすでに設定されている場合に出されます)。

MQeChannelListener

このクラスは、着信の MQSeries Everyplace 論理チャネルのリスナーを作成するために使用します。

パッケージ **com.ibm.mqe**

このクラスは MQe の下位クラスです。

コンストラクター

MQeChannelListener

構文

```
1. public MQeChannelListener( )
2. public MQeChannelListener ( Object listener,
                               String fileType,
                               Object processor )
```

説明 MQeChannelListener オブジェクトを構成します。これは、サーバー (たとえば WebSphere) の制御下で実行されていない場合、着信 MQeChannel 要求を処理するクラスです。このコンストラクターには次の 2 つの形式があります。

1. パラメーターなし。クラスはインスタンス化されますが、アクティブにはなりません。クラスをアクティブにするには、 **activate()** メソッドを呼び出す必要があります。
2. パラメーターあり。次のものを定義します。
   - listen アダプター。たとえば、`Network::80` です。

     注: TCPIP アダプターの場合、`adapter::port_no` は listen を意味します。
   - 着信要求が受け入れられるときに使用されるファイル・タイプ。たとえば、`Network:` です。
   - チャネル要求を処理するクラス・インスタンス。通常これは MQeChannelManager のインスタンスです。

パラメーター

listener MQeAdapter オブジェクト、または着信要求の listen に使用されるファイル記述子ストリングのどちらかを定義するオブジェクト。

fileType MQeAdapter オブジェクトの新しいインスタンスを作成するために使用されるファイル記述子を定義するストリング。そのインスタンスは新しいチャネルのデータの読み取りおよび書き込みに使用されます。

processor チャネルを管理するために使用されるオブジェクトのインスタンス。通常は MQeChannelManager のインスタンスです。

戻り値 なし

例外 なし

例

```
class MySampleClass
  {
  ...
  MQeChannelListener cl = new MQeChannelManager( QNetwork::8080Q,
                                                 QNetwork:Q,
                                                 new MQeChannelManager( ) );
  ...
  }
```

メソッド

メソッド	目的
activate	クラス・コンストラクターによってアクティブにされていない場合、チャネル・リスナーをアクティブにします。
setTimer	このチャネル・リスナーによって受け入れられるチャネルのタイムアウト間隔を設定するために呼び出されます。
stop	新しいインバウンド要求を受け入れるチャネル・リスナーを停止するために呼び出されます。

MQeChannelListener activate

構文

```
public void activate( Object listener,
                      String fileType,
                      Object processor )
```

説明 MQeChannelListener オブジェクトをアクティブにします。通常これは、クラスがパラメーターなしのコンストラクターを使用してインスタンス化される場合にのみ使用されます。パラメーターは以下のものを定義します。

- listen アダプター。たとえば、`Network::80` です。

  注: TCPIP アダプターの場合、`adapter::port_no` は listen を意味します。
- 着信要求が受け入れられるときに使用されるファイル・タイプ。たとえば、`Network:` です。
- チャネル要求を処理するクラス・インスタンス。通常これは MQeChannelManager のインスタンスです。

パラメーター

listener MQeAdapter オブジェクト、または着信要求の listen に使用されるファイル記述子ストリングのどちらかを定義するオブジェクト。

fileType MQeAdapter オブジェクトの新しいインスタンスを作成するために使用されるファイル記述子を定義するストリング。そのインスタンスは新しいチャネルのデータの読み取りおよび書き込みに使用されます。

processor チャネルを管理するために使用されるオブジェクトのインスタンス。通常は MQeChannelManager のインスタンスです。

戻り値 なし

例外 なし

例

```
class MySampleClass
  {
  ...
  MQeChannelListener cl = new MQeChannelManager( );
  cl.activate( QNetwork::8080Q, QNetwork:Q, new MQeChannelManager( ) );
  ...
  }
```

MQeChannelListener setTimer

構文

```
public void setTimer( int interval ) throws Exception
```

説明 このメソッドは、このチャネル・リスナーによって受け入れられるチャネルのチャネル・タイムアウト間隔を設定するために使用されます。

パラメーター

interval 希望するタイムアウト間隔の整数による秒数。

戻り値 なし

例外

MQeException チャネルが無効であるか許可されていません

IOException 入出力操作が失敗しました

例

```
class MySampleClass extends MQe
  {
  MQeChannelListener cl = new MQeChannelManager( "Network::8080",
                                                 "Network:",
                                                  new MQeChannelManager( ) );
    ...
```

```
cl.setTimer( 300 );
  ...
...
}
```

MQeChannelListener stop

構文

```
public void stop( )
```

説明 新しいチャネル要求を受け入れるチャネル・リスナーを停止するために使用されます。

パラメーター

なし

戻り値 なし

例外 なし

例

```
class MySampleClass
  {
  MQeChannelListener cl = new MQeChannelManager( "Network::8080",
                                                 "Network:",
                                                  new MQeChannelManager( ) );
    ...
  cl.stop( );
  ...
  }
```

MQeChannelManager

このクラスは、MQSeries Everyplace 論理チャネルのマネージャーを作成するために使用します。

パッケージ

com.ibm.mqe

このクラスは、MQe の下位クラスです。

コンストラクター

MQeChannelManager

構文

```
public MQeChannelManager( )
```

説明 MQeChannelManager オブジェクトを構成します。

パラメーター

なし

戻り値 なし

例外 なし

例

```
class MySampleClass
  {
  ...
    MQeChannelManager cm = new MQeChannelManager( );
  ...
  }
```

メソッド

メソッド	目的
getGlobalHashtable	共用オブジェクトを保持するために使用されるハッシュ・テーブルへの参照を取得するために呼び出されます。
mapDestination	ある宛先から別の宛先への転送を設定するために呼び出されます。
numberOfChannels	現在アクティブな論理チャネルの数を取得するために呼び出されます。
process	MQSeries Everyplace 論理チャネルに受け取られ、また送られたデータ (バイト) を処理するために呼び出されます。
timeOut	指定の間隔を超えて使用されない場合に論理チャネルを強制的にタイムアウトにするために呼び出されます。

メソッド	目的
totalNumberOfChannels	チャネル・マネージャーがアクティブになってから使用されたチャネルの合計数を取得するために呼び出されます。

MQeChannelManager getGlobalHashtable

構文

```
public Hashtable getGlobalHashTable( )
```

説明

チャネル・マネージャーのこのインスタンスに属するグローバル・ハッシュ・テーブルを戻します。このテーブルを使用して、チャネル間で情報を保持することができます。

パラメーター

なし

戻り値 なし

例外 なし

例

```
class MySampleClass
  {
  try
    {
    MQeChannelManager cm = new MQeChannelManager( );
    Hashtable table = cm.getGlobalHashtable( );
    ...
    }
  catch ( Exception e )
    {
    }
  ...
  }
```

MQeChannelManager mapDestination

構文

```
public void mapDestination(String destination,
                           String newDestination
```

説明 このメソッドは、*destination* から *newDestination* への経路を設定するために使用されます。

パラメーター

destination 再マップされる宛先を定義するストリング

newDestination 新しい宛先を定義するストリング

戻り値 なし

例外 なし

例

```
class MySampleClass
  {
  try
    {
    MQeChannelManager cm = new MQeChannelManager( );
    cm.mapDestination( "One", "Two" );
    ...
    }
  catch ( Exception e )
    {
    }
  ...
  }
```

MQeChannelManager numberOfChannels

構文

```
public int numberOfChannels( int newLimit )
```

説明 このメソッドは、現在アクティブなチャネルの数を戻します。

パラメーター

newLimit このチャネル・マネージャーによって許可されるチャネルの新規の最大数。値 0 は制限なしを意味します。

戻り値 現在のチャネル数の整数値。

例外 なし

例

```
  ...
    MQeChannelManager cm = new MQeChannelManager( );
    int count = cm.numberOfChannels( 0 );
  ...
  ...
```

MQeChannelManager process

構文

1. `public void process( MqeAdapter adapter ) throws Exception`
2. `public void process( MqeAdapter adapter,`
   `byte data[] ) throws Exception`

説明 **process()** メソッドには 2 つの形式があります。

1. MQeAdapter オブジェクトだけをパラメーター指定する。これは、論理チャネルに渡されるデータを読み取るために使用されます。

2. MQeAdapter (またはヌル) とバイトの配列。配列には、論理チャネルによって処理されるデータが入っています。

パラメーター

adapter 入出力操作に使用される MQeAdapter オブジェクト

data 処理されるデータの入ったバイト配列

戻り値 なし

例外

MQeException チャネルが無効であるか許可されていません

data 処理されるデータの入ったバイト配列

例

```
class MySampleClass extends MQe
  {
  try
    {
    MQeChannelManager cm = new MQeChannelManager( );
    ...
    cm.process( null, data );
    ...
    }
  catch ( Exception e )
    {
    }
  ...
  }
```

MQeChannelManager timeOut

構文

1. `public void timeOut( long age )`
2. `public void timeOut( MQeChannel channel,long age )`

説明 このメソッドは、すべてのチャネルまたはある特定のチャネルが使用されないまま *age* ミリ秒を経過していないかどうかを検査するために使用されます。この時間を超えるチャネルはすべてクローズされます。

パラメーター

age ミリ秒での間隔。チャネルが使用されないままこの間隔を経過した場合は、タイムアウトと見なされ、クローズされます。

channel タイムアウトとなっていないかどうか検査される特定の MQSeries Everyplace 論理チャネル。

戻り値 なし

例外 なし

例

```
...
cm.timeOut( 30 * 60 * 1000 );
...
```

MQeChannelManager totalNumberOfChannels

構文

```
public long totalNumberOfChannels( )
```

説明 このメソッドは、チャネル・マネージャーがアクティブになってから使用されたチャネルの合計数を戻します。

パラメーター

なし

戻り値 チャネルの合計数の long 整数値。

例外 なし

例

```
  MQeChannelManager cm = new MQeChannelManager( );

  long count = cm.totalNumberOfChannels( );
...
...
```

MQeEnumeration

このクラスは、MQSeries Everyplace メッセージ・オブジェクトの集合を保持するために使用されます。これにより、メッセージは Java Enumeration クラスと同じ方式で列挙されます。

パッケージ　　**com.ibm.mqe**

java.util.Enumeration をインプリメントします。

メソッド

メソッド	目的
getLockId	メッセージのこのグループに関連づけられる *lockID* (存在する場合) を戻します。
getNextMessage	列挙の中の次のメッセージを戻します。
getQueueManagerName	キューを所有するキュー・マネージャーの名前を戻します。列挙内のメッセージはそのキューからブラウズされます。
getQueueName	列挙内のメッセージのブラウズ元となるキューの名前を戻します。

MQeEnumeration getLockId

構文

```
public long getLockId()
```

説明　この列挙内のメッセージのグループに関連づけられている *lockID* がある場合、このメソッドによって戻されます。この列挙が **browseMessagesAndLock** 操作の結果である場合、 *lockID* だけが設定されます。そうでない場合、このメソッドはダミー値 ″-1″ を戻します。

パラメーター

なし

戻り値　この列挙内のメッセージのグループの *lockID* が入った long 値。

例外　なし

例

```
class MyMQeApplication
{
  ...
  /* Lock all msgs on this queue */
  MQeEnumeration msgEnum = QMgr.browseMessagesAndLock( null, "MyQueue",
                                                       null, null, 0, false );
  long lockId = msgEnum.getLockId(); /* get the Lock Id */
 ...
}
```

MQeEnumeration getNextMessage

構文

```
public MQeMsgObject getNextMessage( MQeAttribute attribute,
                                    long confirmId ) throws Exception
```

説明 このメソッドは、列挙の中の次のメッセージを戻します。ただし、このメソッドの振る舞いは、この列挙を作成したブラウズ要求の *justID* パラメーターに応じて異なります。 *justID* パラメーターは、ブラウズによって一致するメッセージの固有 ID フィールドだけを列挙に入れるか、各メッセージのすべてのフィールドを列挙に入れるかを決定します。

ブラウズ要求の *justUID* パラメーターが false に設定されている場合、このメソッドは列挙内の次のメッセージを戻します (この場合、**nextElement()** メソッドの働きと同じです)。

ブラウズ要求の *justUID* パラメーターが true に設定されている場合、このメソッドはターゲット・キューに get message コマンドを出すことによってメッセージを戻します。これにより、ターゲット・キューからメッセージを除去します。

nextElement() メソッド (java.util.Enumeration から継承) を使用すると、ターゲット・キューから除去せずにメッセージを戻すことができます。

パラメーター

attribute メッセージ・レベルのセキュリティーを提供するために使用する MQeAttribute オブジェクト。属性は、このメソッドで戻されるメッセージに付加された属性と一致しなければなりません。このようになっていないと、メッセージが消失する可能性があります。

confirmId 保証されたメッセージ送達を使用するかどうかを示す long 値。非ゼロ値の場合、メッセージはターゲット・キューから除去されません。これが行われるのは以降の確認フローのときです。ゼロの値の場合、メッセージはターゲット・キューから即時に除去されます。

戻り値 列挙内の次のエレメントの入った MQeMsgObject

例外 Except_NotSupported

例

```
class MyMQeApplication
{
  ...
  MQeEnumeration msgEnum = null;
  msgEnum = qmgr.browseMessages( "RemoteQMgr", "RemoteQueue", null, null,
                                 false );
  while( msgEnum.hasMoreElements() )
  {
    /* get message */
```

```
    MQeMsgObject msg = msgEnum.getNextMessage( null, MQe.uniqueValue() );
    /* confirm get */
    qmgr.confirmGetMessage( msgEnum.getQueueManagerName(),
                            msgEnum.getQueueName(),
                            msg.getMsgUIDFields() );
  }
  ...
}
```

MQeEnumeration getQueueManagerName

構文

```
public String getQueueManagerName()
```

説明 このメソッドは、キューを所有するキュー・マネージャーの名前を戻します。列挙内のメッセージはそのキューからブラウズされます。

パラメーター

なし

戻り値 これらのメッセージのブラウズ元となるキューを所有するキュー・マネージャーの名前の入ったストリング。

例外 なし

例

```
cclass MyMQeApplication
{
  ...
  MQeEnumeration msgEnum = null;
  msgEnum = qmgr.browseMessages( "RemoteQMgr", "RemoteQueue", null, null,
                                 false );
  while( msgEnum.hasMoreElements() )
  {
    /* get message */
    MQeMsgObject msg = msgEnum.getNextMessage( null, MQe.uniqueValue() );
    /* confirm get */
    qmgr.confirmGetMessage( msgEnum.getQueueManagerName(),
                            msgEnum.getQueueName(), msg.getMsgUIDFields() );  }
  ...
}
```

関連する関数

getQueueName()

MQeEnumeration getQueueName

構文 public String getQueueName()

説明 このメソッドは、列挙内のメッセージのブラウズ元となるキューの名前を戻します。

パラメーター

なし

戻り値 これらのメッセージのブラウズ元となるキューの名前の入ったストリング。

例外　なし

例

```
class MyMQeApplication
{
  ...
  MQeEnumeration msgEnum = null;
  msgEnum = qmgr.browseMessages( "RemoteQMgr", "RemoteQueue", null, null,
                                 false );
  while( msgEnum.hasMoreElements() )
  {
    /* get message */
    MQeMsgObject msg = msgEnum.getNextMessage( null, MQe.uniqueValue() );
    /* confirm get */
    qmgr.confirmGetMessage( msgEnum.getQueueManagerName(),
                            msgEnum.getQueueName(), msg.getMsgUIDFields() );  }
  ...
}
```

関連する関数

getQueueManagerName

MQeException

このクラスは MQeException オブジェクトを作成するときに使用します。

パッケージ **com.ibm.mqe**

このクラスは MQe の下位クラスです。

コンストラクター

MQeException

構文

```
1. public MQeException( )
2. public MQeException( int codeValue  )
3. public MQeException( String errorMsg )
4. public MQeException( int codeValue, String errorMsg )
```

説明 MQeException オブジェクトを構成します。このコンストラクターには次の 5 つの形式があります。

1. *codeValue* が 0 で、エラー・メッセージのないオブジェクトを作成します。
2. *codeValue* が指定値で、エラー・メッセージのないオブジェクトを作成します。
3. *codeValue* が 0 で、エラー・メッセージのあるオブジェクトを作成します。
4.
   a. *codeValue* が指定値で、エラー・メッセージのあるオブジェクトを作成します。
   b. *codeValue* が指定値で、エラー・メッセージがあって、データを組み込んだ (隠した) オブジェクトを作成します。

codeValue パラメーターの値は、 MQe クラスで定義されている定数の 1 つでなければなりません。たとえば、MQe.Except_NotFound。

パラメーター

codeValue 整数値。通常は MQe.Except_... 定数の 1 つです。

errorMsg 例外に関連したストリング。例外が発生すると表示されます。

戻り値 なし

例外 なし

例

```
class MySampleClass
   {
   ...
   if ( data == null )
     throw new MQeException( MQe.Except_Data, "Data missing" );
   ...
   ...
   }
```

メソッド

メソッド	目的
code	例外の整数値を戻します。

MQeException code

構文

```
public int code( )
```

説明 このメソッドは、MQeException のコード値を抽出します。例外が発生したときに設定された値です。

パラメーター

なし

戻り値 整数

例外 なし

例

```
 class MySampleClass
   {
   ...
   try
     {
     ...
     }
   catch ( Exception e )
     {
     if ( e instanceof MQeException )
       switch (((MQeException) e).code( ) )
         {
         case MQe.Except_Data:
           System.err.println( "Data format error" );
           break;
         case MQe. Except_NotFound:
           System.err.println( "Data not specified" );
           break;
         }
```

```
  else
    System.err.println( "Error:" + e.toString( ) );
...
}
```

MQeFields

このクラスは、基本的な MQeFields オブジェクトを作成するために使用します。このオブジェクトを使用して、さまざまなデータ項目を保持し、これらのフィールド項目をバイト配列にダンプしたり、バイト配列から復元したりするメカニズムを提供します。

フィールド項目は、 MQeFields オブジェクトに追加されるときに、文字による名前が割り当てられます。この名前は、以下の条件を満たしていなければなりません。

- 長さが 1 文字以上である
- ASCII 文字セット (つまり、20 より大きく 128 より小さい値の文字) に準拠している
- {}[]#():;,'"= という文字はいずれも使用できない

注: これらのルールは必須ではありませんが、従わない場合は結果は予期できないものになります。

パッケージ **com.ibm.mqe**

このクラスは、MQe の下位クラスです。

コンストラクター

コンストラクター	目的
MQeFields	MQeFields オブジェクトを作成して初期化します。

MQeFields

構文

1. `public MQeFields( )`
2. `public MQeFields( byte data[] )`

説明 このコンストラクターは、MQeFields オブジェクトを作成して初期化します。このコンストラクターには次の 2 つの形式があります。

1. パラメーターなし。この場合、空の MQeFields オブジェクトを構成します。
2. バイト配列付き。この場合、指定されたバイト配列から MQeFields オブジェクトを復元します。

   注: 各オブジェクトは同じタイプでなければなりません。

パラメーター

data ダンプされた MQeFields オブジェクトの入ったバイト配列

戻り値 なし

例外

MQeException	Except_data, ″data:xxxx″
	Except_Type, ″Type: aaaa - bbbb″

例

```
class MyApplication
   {
   ...
   MQeFields fields = new MQeFields( );
   ...
   ...
   }
```

メソッド

メソッド	目的
contains	オブジェクト内にフィールドが存在するかどうかを検査します。
copy	ある MQeFields オブジェクトから別の同オブジェクトに 1 つのフィールドまたは一連のフィールドをコピーします。
dataType	オブジェクト内のフィールドのデータ・タイプを決定します。
delete	オブジェクトからフィールドを除去します。
dump	メッセージ・オブジェクトの内容をバイト配列にダンプします。
dumpedType	ダンプされた MQeFields オブジェクトのオブジェクト・タイプを戻します。
dumpToString	MQeFields オブジェクトの内容を人が読める表現にします。
equals	別の MQeFields オブジェクトとの等価性テストを実施します。
fields	オブジェクト内のすべてのフィールドのリストを戻します。
getArrayLength	フィールドの動的配列の length 値を抽出します。
getArrayOfByte	バイトの固定サイズ配列を抽出します。
getArrayOfDouble	ダブル・サイズ浮動小数点数の固定サイズ配列を抽出します。
getArrayOfFloat	浮動サイズ浮動小数点数の固定サイズ配列を抽出します。
getArrayOfInt	int サイズ整数の固定サイズ配列を抽出します。
getArrayOfLong	long サイズ整数の固定サイズ配列を抽出します。
getArrayOfShort	short サイズ整数の固定サイズ配列を抽出します。
getAscii	ASCII ストリングを抽出します。
getAsciiArray	ストリングの ASCII 配列を抽出します。
getAttribute	現在の属性オブジェクト参照を抽出します。
getBoolean	ブール値またはヌルを抽出します。
getByte	バイト値を抽出します。
getByteArray	バイト値の動的サイズ配列を抽出します。
getDouble	倍精度浮動小数点値を抽出します。

メソッド	目的
getDoubleArray	倍精度浮動小数点値の動的サイズ配列を抽出します。
getFields	組み込み MQeFields オブジェクトを抽出します。
getFieldsArray	MQeFields オブジェクトの動的サイズ配列を抽出します。
getFloat	浮動値を抽出します。
getFloat	浮動値を抽出します。
getInt	整数を抽出します。
getIntArray	動的サイズ整数配列を抽出します。
getLong	long 整数を抽出します。
getLongArray	動的サイズの long 整数配列を抽出します。
getShort	short 整数を抽出します。
getShortArray	short 整数配列を抽出します。
getUnicode	Unicode ストリングを抽出します。
getUnicodeArray	Unicode ストリングの動的サイズ配列を抽出します。
hide	フィールドが等価検査に使用されないようにします。
putArrayLength	フィールドの動的配列の length 値を設定します。
putArrayOfByte	バイトの固定サイズ配列を設定します。
putArrayOfDouble	ダブル・サイズ浮動小数点数の固定サイズ配列を設定します。
putArrayOfFloat	浮動サイズ浮動小数点数の固定サイズ配列を設定します。
putArrayOfInt	int サイズ整数の固定サイズ配列を設定します。
putArrayOfLong	long サイズ整数の固定サイズ配列を設定します。
putArrayOfShort	short サイズ整数の固定サイズ配列を設定します。
putAscii	ASCII 文字の入ったストリングを設定します。
putAsciiArray	ASCII 文字の入ったストリングの動的サイズ配列を設定します。
putBoolean	ブール値を設定します。
putByte	バイトからのデータをメッセージに設定します。
putByteArray	バイト値の動的サイズ配列を設定します。
putDouble	倍精度浮動小数点値を設定します。
putDoubleArray	動的サイズの倍精度浮動小数点配列を設定します。
putFields	組み込み MQeFields オブジェクトを設定します。
putfFieldsArray	MQeFields オブジェクトの配列を設定します。
putFloat	浮動値を設定します。
putFloatArray	動的サイズ浮動配列を設定します。
putInt	整数を設定します。
putIntArray	動的サイズ整数配列を設定します。
putLong	long 整数を設定します。
putLongArray	動的サイズ long 整数配列を設定します。
putShort	short 整数を設定します。

メソッド	目的
putShortArray	動的サイズ short 整数配列を設定します。
putUnicode	Unicode 文字の入ったストリングを設定します。
putUnicodeArray	Unicode 文字の入ったストリングの動的サイズ配列を設定します。
rename	MQeFields オブジェクトに保持されている項目の名前を変更します。
restore	**dump()** メソッドによって生成されるバイト配列から MQeFields オブジェクトの内容を復元します。
restoreFromFile	2 進数または定様式 ASCII 形式のファイルから MQeFields オブジェクトの内容を復元します。
restoreFromString	ASCII ストリング (通常は **dumpToString()** メソッド呼び出しによって生成される) から MQeFields オブジェクトの内容を復元します。
setAttribute	MQeFields オブジェクトに属性オブジェクトを割り当てます。
updateValue	MQeFields オブジェクト内の整数型の値を更新 (増分または減分) します。

MQeFields contains

構文

```
public boolean contains( String item )
```

説明 このメソッドは、MQeFields オブジェクト内にフィールドが存在するかどうかを検査します。

パラメーター

item 検査される項目の名前。

戻り値

true フィールドが検出されました。

false フィールドは検出されませんでした。

例外 なし

例

```
class MyApplication
   {
   ...
   MQeFields msg = new MQeFields( );
   msg.putAscii("Data", "This is some data" );
   ...
   if ( msg.contains( "Data" ) )
     ...
   ...
   }
```

MQeFields copy

構文

```
1. public void copy( MQeFields from,
                     boolean  replace )
2. public void copy( MQeFields  from,
                     boolean  replace,
                     String   item )
```

説明 このメソッドは、ある MQeFields オブジェクトから別の同オブジェクトに 1 つのフィールド (またはすべてのフィールド) に対する参照をコピーします。2 つの形式があります。

1. すべてのフィールドをコピーします。
2. 個々のフィールドをコピーします。

ブール値 *replace* が `false` に設定されている場合は、ターゲットである MQeFields オブジェクト内にフィールドがすでに存在していると例外が出され、 `true` に設定されている場合は、既存の値を置き換えます。

パラメーター

from	データのソースとして使用される MQeFields オブジェクト。
replace	フィールドを置き換えるかどうかを決めるブール値。
item	コピーされる単一フィールドの名前。

戻り値 なし

例外

MQeException	`Except_Duplicate, "Duplicate: aaaa"`

例

```
class MyApplication
  {
  ...
  MQeFields fields1 = new MQeFields( );
  fields1.putAscii("data", "This issome data" );
  ...
  MQeFields fields2 = new MQeFields( );
  fields2.copy(fields1, true, "data" )
  ...
  ...
  }
```

MQeFields dataType

構文

```
public char dataType( String item )
```

説明 このメソッドは、 MQeFields オブジェクト内のフィールドのデータ・タイプを戻します。

パラメーター

item 検査される項目の名前。

戻り値 フィールドのデータ・タイプを表す文字値。 MQeFields に事前定義されているデータ・タイプは以下のとおりです。

```
public    final static char    TypeUnTyped
public    final static char    TypeAscii
public    final static char    TypeUnicode
public    final static char    TypeBoolean
public    final static char    TypeByte
public    final static char    TypeShort
public    final static char    TypeInt
public    final static char    TypeLong
public    final static char    TypeFloat
public    final static char    TypeDouble
public    final static char    TypeArrayElements
public    final static char    TypeFields
```

例外

MQeException さまざまなものがあります。

例

```
class MyApplication
  {
  ...
  MQeFieldsmsg = new MQeFields( );
  msg.putAscii("Data", "This is some data" );
  ...
  if ( msg.dataType( "Data" ) ==TypeAscii) {
  ...
  }
```

MQeFields delete

構文

```
public void delete( String item )
```

説明 このメソッドは、MQeFields オブジェクトから既存のフィールドを削除します。

パラメーター

item 除去される項目の名前。

戻り値

例外 なし

例

```
class MyApplication
  {
  ...
```

```
MQeFields msg = new MQeFields( );
msg.putAscii("Data", "This is some data" );
...
msg.delete( "Data" );
...
}
```

MQeFields dump

構文

1. `public byte[] dump( ) throws Exception`

2. `public byte[] dump( boolean allowXor ) throws Exception`

説明 このメソッドは、**restore()** メソッドを使用して復元できるよう、この MQeFields オブジェクトの内容をバイト配列にダンプします。このメソッドには 2 つの形式があります。

1. パラメーターなし。
2. *allowXor* 付き。これが `false` に設定されている場合、MQeFields オブジェクトはバイト配列にダンプされます。 *allowXor* を `true` に設定すると、各フィールドは、圧縮率を高めるために、`0x00` 値を持つバイトの数を増やそうとして、以前のバージョン (内部に保持されている) で XOR 処理されます。

インテリジェント・フィールド・オブジェクト、つまり、プログラム・ロジックを持つフィールドを作成する場合、このロジックは、 **dump()** および **restore()** メソッドでアクティブにしておく必要があります。たとえば、MQeFields オブジェクトをダンプする直前に、データベース照会を発行して最新のデータを取得したり、復元の直後に、データをデータベースに自動的に保管したりできます。

パラメーター

allowXor ブール式。 `true` はフィールドを XOR 処理することを暗黙指定し、 `false` はフィールドを XOR 処理しないことを暗黙指定します。

戻り値 なし

例外

MQeException さまざまなものがあります。

例

```
class MyApplication
  {
  ...
  MQeFields msg = new MQeFields( );
  msg.putAscii( "Data", "This is some data" );
```

```
...
byte dumpData[] = msg.dump( );
...
}
```

MQeFields ダンプ・データ形式: MQSeries Everyplace 環境間で送信されるデータは以下のレイアウトでエンコードされます。

```
{Length  Identifier  Fence  {Data}}  {Length  Identifier  Fence  {Data}} { ...}
```

ここで、

Length

1 〜 4 までのバイト変数。 length は以下の方法でエンコードされます。

最初のバイトには最初の 2 ビットが予約されており、 length フィールドの長さとして使用されます。

00 = 長さに 1 バイト (6 ビット = 0-63)

01 = 長さに 2 バイト (14 ビット = 0-16,383)

10 = 長さに 3 バイト (22 ビット = 0-4,194,303)

11 = 長さに 4 バイト (30 ビット = 0-1,073,741,823)

Identifier

バイトの可変長ストリング (各バイト値は 0x80 未満でなければならない) で、通常、これは ASCII ストリングとなります。 ID の終わりは、0xC0 ビットの設定されているバイトが検出されたときに決まります。この ID には以下に示す制約事項があります。

- 長さが 1 文字以上である
- ASCII 文字セット (つまり、20 より大きく 128 より小さい値) に準拠している
- {}[]#():;,'=" という文字はいずれも使用できない

Fence

ID とオプションのデータ項目の境界を区切る特別なバイト。このバイトは、以下の例に示すようにデータ項目のデータ・タイプを入れるために使用されます。

```
/* Field mask values */
public    final static char    TypeFenceMask      = 0x00C0;
public    final static char    TypeHidden         = 0x0020;
public    final static char    TypeModifier       = 0x0010;
/* Field data types */
public    final static char    TypeUnTyped        = 0x0000 | TypeFenceMask;
public    final static char    TypeAscii          = 0x0001 | TypeFenceMask;
public    final static char    TypeUnicode        = 0x0002 | TypeFenceMask;
public    final static char    TypeBoolean        = 0x0003 | TypeFenceMask;
public    final static char    TypeByte           = 0x0004 | TypeFenceMask;
public    final static char    TypeShort          = 0x0005 | TypeFenceMask;
public    final static char    TypeInt            = 0x0006 | TypeFenceMask;
public    final static char    TypeLong           = 0x0007 | TypeFenceMask;
```

```
public    final static char    TypeFloat          = 0x0008 | TypeFenceMask;
public    final static char    TypeDouble         = 0x0009 | TypeFenceMask;
public    final static char    TypeArrayElements  = 0x000A | TypeFenceMask;
public    final static char    TypeFields         = 0x000B | TypeFenceMask;
```

データ・ストリーム内の項目の順番は重要ではありません。

```
08 5349 C7 1122334455 |02 44 D3 |03 5349 D6 |4603 534443 C4
6E01534FE32054E16....
```

送信されるバイト数を少なくする: このデータ構造を使用してバイト・ストリームに保管するには以下のようにします。

- 最初の長さバイトの 2 ビットを予約することによって可変長の長さを可能にします。可変長の長さコード (1 〜 4 バイト)、たとえば必須の長さバイトだけが送信されます。
- 整数値の先頭の 0x00 および 0xFF は出力ストリームには書き出されません。値が 0 または -1 の場合、データ・バイトは何も送信されません。
- 終了を受け取るときに、すべてのデータ項目のタイプが決められ、タイプが検査されます。
- 引き続きヌル項目が送信されている (データ・タイプ付きで) 場合、終了を受け取るときに項目の存在が検査されます。
- 3 つの別個の関数のために分離バイトを使用します。
  1. ID を区切る
  2. データ・タイプを定義する
  3. データが次のような条件であることを定義する
  - ヌル (データ・バイトなし)
  - 正または負 (0 または -1 の場合、データ・バイトは送信されないことになる)
  - ブールの true または false (データ・バイトは送信されない)

注: さらにサイズを少なくするにはデータの圧縮を行います。圧縮機能は、通常、 0x00 バイトの繰り返しを生成して、以前のバイト・ストリームで XOR 処理を実行することによって促進できますが、これらのフィールドの特性が多様で、フィールドの順序が変わる可能性があるため、単純な XOR では期待する成果が出ない場合があります。しかし、「インテリジェントな」XOR はフィールドごとに機能して、0x00 バイトの繰り返しを生成し、圧縮機能を補助します。

MQeFields dumpToString

構文

```
public String dumpToString( String template )
```

説明 このメソッドは、MQeFields オブジェクトを人間が読める形式でダンプして、データをストリングとして戻します。

パラメーター

template 出力を形式設定するときに使用されるストリング・テンプレート。テンプレートには 3 つの挿入シーケンス '#n' があります。つまり、以下のようになります。

- "#0" データ・タイプ用
- "#1" フィールド名用
- "#2" フィールド値用

例:

```
"Sample template -Name=#1, Type=#0, Value=#2"
```

戻り値 MQeFields オブジェクトの表記の入ったストリング。

例外 さまざまな変換の例外があります。

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  fields.putBoolean( "tb", true );
  ...
  fields.putLong( "ml", -1 );
  System.out.println( fields.dumpToString( "Test1.obj  (#0)¥t#1¥t=#2¥r¥n" ) );
  ...
  }
```

dumpToString 呼び出しから出力の例:

```
Test1.obj  (long)     la   =[2] { 0000000000000001, FFFFFFFFFFFFFFFE }
Test1.obj  (boolean)  tb   =true Test1.obj  (byte) ba =[5] { 01, FE, FD, 04, 05 }
Test1.obj  (long)     pl   =101
Test1.obj  (ascii)     A  =Ascii string
Test1.obj  (ascii)   nA  =null
Test1.obj  (unicode)   U  =Unicode string
Test1.obj  (byte)     mb  =[1] { FE }
Test1.obj  (int)       i   =1
Test1.obj  (byte)    pb  =[1] { 02 }
Test1.obj  (boolean)  fb  =false
Test1.obj  (short)   ms  =-1
Test1.obj  (short)   sa  =[5] { 0001, FFFE, FFFD, 0004, 0005}
Test1.obj  (short)   ps  =0
Test1.obj  (int)     ia  =[3] { 00000001, FFFFFFFE, FFFFFFFD }
Test1.obj  (long)    ml  =-1
```

MQeFields dumpedType

構文

```
public static String dumpedType( byte data[] ) throws Exception
```

説明 このメソッドは、ダンプされたオブジェクトのクラス名の入ったストリングを戻します。

パラメーター

data MQeFields オブジェクトのダンプの入ったバイト配列

戻り値 ダンプされたオブジェクトのクラス名の入ったストリング。

例外

MQeException	Except_Data, ″Data:aaa″

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  fields.putAscii( "Data", "This is some data" );
  byte dumpdata[] = fields.dump(fields.dump( );
 ...
 String ObjType = fields.dumpedType(objType = MQeFields.dumpedType( dumpdata );
 ...
 }
```

MQeFields equals

構文

```
public boolean equals( MQeFields match ) throws Exception
```

説明

デフォルト・メソッドは、パラメーターとして MQeFields (またはその下位) を必要とします。パラメーター・オブジェクトの各フィールドは、 MQeFields の突き合わせフィールドとの等価性について検査されます。

異なるタイプの等価性検査を提供するには、このメソッドを変更します。

パラメーター

match	比較に使用される項目の入った MQeFields オブジェクト。

戻り値 一致する場合は `true`、そうでない場合は `false`。

例外

MQeException	Except_Type,″wrong field type″
MQeException	Except_NotFound, item + ″ not found″

例

```
class MyApplication
  {
  ...
   MQeFields fields = new MQeFields( );
  fields.putAscii( "Data1", "This is some data" );
  fields.putAscii( "Data2", "This is more data" );
  ...
  MQeFields test = new MQeFields( );
  test.putAscii( "Data1", "This is some data" );
  ...
  if ( fields.equals( test ) )
```

```
  ...
else
  ...
}
```

MQeFields fields

構文

```
public Enumeration fields( )
```

説明 このメソッドは、オブジェクト内のすべてのフィールド名の入った列挙型オブジェクトを戻します。

パラメーター

なし

戻り値 フィールド名の入った列挙型オブジェクト。

例外

MQeException	Except_Type, ″wrong field type″
MQeException	Except_NotFound, item + ″ not found″

例

```
class MyApplication
  {
  ...
   MQeFields fields = new MQeFields( );
  fields.putAscii( "data", "This is some data" );
  ...
  Enumeration names = fields.fields( );
  ...
  }
```

MQeFields getArrayLength

構文

```
 public int getArrayLength( String item ) throws Exception
```

説明 これは、指定項目の動的配列長さを抽出します。データがない場合、またはデータ・タイプが間違っている場合に、例外が起こります。

パラメーター

item	検索される項目の名前。

戻り値 メッセージからの ASCII データの入ったストリングの配列。

例外

MQeException	Except_Type, ″wrong field type″
	Except_NotFound, item + ″ not found″

例

```
class MyApplication
   {
   ...
   MQeFields fields = new MQeFields( dumpData );
   ...
   int numElements = fields.getArrayLength( "Data" );
   ...
   }
```

MQeFields getArrayOfByte

構文

```
public byte[] getArrayOfByte( String item ) throws Exception
```

説明 これは、MQeFields オブジェクトからバイト・データの配列を抽出します。データがない場合、またはデータ・タイプが間違っている場合に、例外が起こります。

パラメーター

item 検索される項目の名前。

戻り値 メッセージからのデータの入ったバイト配列。

例外

MQeException	Except_Type, ″wrong field type″
	Except_NotFound, item + ″ not found″

例

```
class MyApplication
   {
   ...
   MQeFields fields = new MQeFields( );
   fields.restore( dumpData );
   ...
   byte data[] = fields.getArrayOfByte( "Data" );
   ...
   }
```

MQeFields getArrayOfDouble

構文

```
public double[] getArrayOfDouble( String item ) throws Exception
```

説明 これは、MQeFields オブジェクトから倍精度浮動小数点数の配列を抽出します。データがない場合、またはデータ・タイプが間違っている場合に、例外が起こります。

パラメーター

item 検索される項目の名前。

戻り値 ダブル値の配列。

例外

MQeException	Except_Type, ″wrong field type″
	Except_NotFound, item + ″ not found″

例

```
class MyApplication
 {
  ...
  MQeFields fields = new MQeFields( );
  fields.restore( dumpData );
  ...
  double data[] = fields.getArrayOfDouble( "Data" );
  ...
 }
```

MQeFields getArrayOfFloat

構文

```
public float[] getArrayOfFloat( String item ) throws Exception
```

説明 これは、MQeFields オブジェクトから浮動小数点数の配列を抽出します。データがない場合、またはデータ・タイプが間違っている場合に、例外が起こります。

パラメーター

item 検索される項目の名前。

戻り値 浮動値の配列。

例外

MQeException	Except_Type, ″wrong field type″
	Except_NotFound, item + ″ not found″

例

```
class MyApplication
  {
  ...

  MQeFields fields = new MQeFields( );
  fields.restore( dumpData );
  ...
  float data[] = fields.getArrayOfFloat( "Data" );
  ...
  }
```

MQeFields getArrayOfInt

構文

```
public int[] getArrayOfInt( String item ) throws Exception
```

説明 これは、MQeFields オブジェクトから int 型の長さの整数値の配列を抽出します。データがない場合、またはデータ・タイプが間違っている場合に、例外が起こります。

パラメーター

item 検索される項目の名前。

戻り値 int 値の配列

例外

MQeException	Except_Type, ″wrong field type″
	Except_NotFound, item + ″ not found″

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  fields.restore( dumpData );
  ...
  int data[] = fields.getArrayOfInt( "Data" );
  ...
  }
```

MQeFields getArrayOfLong

構文

```
public long[] getArrayOfLong( String item ) throws Exception
```

説明 これは、MQeFields オブジェクトから long 型の長さの整数値の配列を抽出します。データがない場合、またはデータ・タイプが間違っている場合に、例外が起こります。

パラメーター

item 検索される項目の名前。

戻り値 long 値の配列

例外

MQeException	Except_Type, ″wrong field type″
	Except_NotFound, item + ″ not found″

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  fields.restore( dumpData );
  ...
  long data[] = fields.getArrayOfLong( "Data" );
  ...
  }
```

MQeFields getArrayOfShort

構文

```
public short[] getArrayOfShort( String item ) throws Exception
```

説明 これは、MQeFields オブジェクトから short 型の長さの整数値の配列を抽出します。データがない場合、またはデータ・タイプが間違っている場合に、例外が起こります。

パラメーター

item 検索される項目の名前。

戻り値 short 値の配列

例外

MQeException	Except_Type, ″wrong field type″
	Except_NotFound, item + ″ not found″

例

```
class MyApplication
 {
  ...
  MQeFields fields = new MQeFields( );
  fields.restore( dumpData );
  ...
  short data[] = fields.getArrayOfShort( "Data" );
  ...
 }
```

MQeFields getAscii

構文

```
public String getAscii( String item ) throws Exception
```

説明 これは、MQeFields オブジェクトから ASCII データを抽出し、それをストリングとして戻します。データがない場合、またはデータ・タイプが間違っている場合に、例外が起こります。

注: *item* パラメーターは Java Unicode ストリングであり、 ASCII コード・ページの不変の部分の文字コードだけが含まれる必要があります (20 より

大きく 128 より小さい文字で、{}[]#():;'"= は含みません)。変化する文字コードを渡そうとする場合、これらのコードは異なるコード・ページが使用されているマシン間でデータを処理するためには変換する必要があり、予期しない結果が生じる場合もあります。 MQSeries Everyplace メッセージに変化する文字コードを渡す場合は、 **putArrayOfByte()** メソッドを使用してマシン間の独自のコード・ページ変換を行うか、コード・ページ変換の必要ない **putUnicode()** メソッドを使用することをお勧めします。

パラメーター

item 検索される項目の名前。

戻り値 メッセージからの ASCII データの入ったストリング。

例外

MQeException Except_Type, ″wrong field type″

Except_NotFound, item + ″ not found″

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields();
  fields.restore( dumpData );
  ...
  String data = fields.getAscii( "Data" );
  ...
  }
```

MQeFields getAsciiArray

構文

```
public String[] getAsciiArray( String item ) throws Exception
```

説明 これは、MQeFields オブジェクトから ASCII データ (108ページの『MQeFields getAscii』の注を参照) を抽出し、それをストリングの配列として戻します。データがない場合、またはデータ・タイプが間違っている場合に、例外が起こります。

パラメーター

item 検索される項目の名前。

戻り値 メッセージからの ASCII データの入ったストリングの配列。

例外

MQeException Except_Type, ″wrong field type″

Except_NotFound, item + ″ not found″

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  fields.restore( dumpData );
  ...
  String data[] = fields.getAsciiArray( "Data" );
  ...
  }
```

MQeFields getAttribute

構文

```
public MQeAttribute getAttribute( )
```

説明 このメソッドは、この MQeFields オブジェクトに関連付けられた MQeAdminQueueAdminMsg オブジェクト参照を戻すか、または属性がない場合はヌルを戻します。

パラメーター

なし

戻り値 MQeAdminQueueAdminMsg オブジェクト参照

例外 なし

例

```
class MyApplication
  {
  ...
  MQeAttribute thisAttribute = fields.getAttribute( );
  ...
  }
```

MQeFields getBoolean

構文

```
public boolean getBooean( String item ) throws Exception
```

説明 これは、MQeFields オブジェクトからブール値を抽出します。データがない場合、またはデータ・タイプが間違っている場合に、例外が起こります。

パラメーター

item 検索される項目の名前。

戻り値 `true` または `false` に設定されたブール値

例外

MQeException `Except_Type, ″wrong field type″`

Except_NotFound, item + ″ not found″

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  fields.restore( dumpData );
  ...
  boolean data = fields.getBoolean( "Data" );
  ...
  }
```

MQeFields getByte

構文

```
public byte getByte( String item ) throws Exception
```

説明 これは、MQeFields オブジェクトからバイト・データを抽出します。データがない場合、またはデータ・タイプが間違っている場合に、例外が起こります。

パラメーター

item 検索される項目の名前。

戻り値 フィールドからのデータの入ったバイト

例外

MQeException Except_Type, ″wrong field type″

Except_NotFound, item + ″ not found″

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  fields.restore( dumpData );
  ...
  byte data = fields.getByte( "Data" );
  ...
  }
```

MQeFields getByteArray

構文

```
public byte[] getByteArray( String item ) throws Exception
```

説明 これは、MQeFields オブジェクトからバイト・データを抽出し、それをバイト配列として戻します。データがない場合、またはデータ・タイプが間違っている場合に、例外が起こります。

パラメーター

item 検索される項目の名前。

戻り値 フィールドからのデータの入ったバイト配列。

例外

MQeException	Except_Type, ″wrong field type″
	Except_NotFound, item + ″ not found″

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  fields.restore( dumpData );
  ...
  byte data[] = fields.getByteArray( "Data" );
  ...
  }
```

MQeFields getDouble

構文

```
public double getDouble( String item ) throws Exception
```

説明 これは、MQeFields オブジェクトから 2 倍長の浮動小数点値を抽出します。データがない場合、またはデータ・タイプが間違っている場合に、例外が起こります。

パラメーター

item 検索される項目の名前。

戻り値 フィールドからの値の入った倍の値。

例外

MQeException	Except_Type, ″wrong field type″
	Except_NotFound, item + ″ not found″

例

```
class MyApplication
   {
   ...
   MQeFields fields = new MQeFields( );
   fields.restore( dumpData );
   ...
   double data = fields.getDouble( "Data" );
   ...
   }
```

MQeFields getDoubleArray

構文

```
public byte[] getDoubleArray( String item ) throws Exception
```

説明 これは、MQeFields オブジェクトから 2 倍長の浮動小数点値の動的配列を抽出し、それを 2 倍長の配列として戻します。配列の長さは、この項目の *ArrayLength* 値によって決まります。データがない場合、またはデータ・タイプが間違っている場合に、例外が起こります。

パラメーター

item 検索される項目の名前。

戻り値 フィールドからのデータの入ったバイト配列。

例外

MQeException	Except_Type, ″wrong field type″
	Except_NotFound, item + ″ not found″

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  fields.restore( dumpData );
  ...
  double data[] = fields.getDoubleArray( "Data" );
  ...

  }
```

MQeFields getFields

構文

```
public MQeFields getFields( String item ) throws Exception
```

説明 これは、MQeFields オブジェクトからフィールド項目を抽出します。データがない場合、またはデータ・タイプが間違っている場合に、例外が起こります。

パラメーター

item 検索される項目の名前。

戻り値 フィールドからの値の入った倍の値。

例外

MQeException	Except_Type, ″wrong field type″
	Except_NotFound, item + ″ not found″

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  fields.restore( dumpData );
  ...
  MQeFields data = fields.getFields( "Data" );
  ...
  }
```

MQeFields getFieldsArray

構文

```
public MQeFields[] getFieldsArray( String item ) throws Exception
```

説明 これは、MQeFields オブジェクトからフィールド・オブジェクトの動的配列を抽出し、それを配列として戻します。配列の長さは、この項目の *ArrayLength* 値によって決まります。データがない場合、またはデータ・タイプが間違っている場合に、例外が起こります。

パラメーター

item 検索される項目の名前。

戻り値 MQeFields オブジェクトの入った配列。

例外

MQeException	Except_Type, ″wrong field type″
	Except_NotFound, item + ″ not found″

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  fields.restore( dumpData );
  ...
  MQeFields data[] = fields.getFieldsArray( "Data" );
  ...
  }
```

MQeFields getFloat

構文

```
public float getFloat( String item ) throws Exception
```

説明 これは、MQeFields オブジェクトから浮動値項目を抽出します。データがない場合、またはデータ・タイプが間違っている場合に、例外が起こります。

パラメーター

item 検索される項目の名前。

戻り値 フィールドからの値の入った浮動値。

例外

MQeException	Except_Type, ″wrong field type″
	Except_NotFound, item + ″ not found″

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  fields.restore( dumpData );
  ...
  float data = fields.getFloat( "Data" );
  ...
  }
```

MQeFields getFloatArray

構文

```
public float[] getFloatArray( String item ) throws Exception
```

説明 これは、MQeFields オブジェクトから浮動値の動的配列を抽出し、それを配列として戻します。配列の長さは、この項目の *ArrayLength* 値によって決まります。データがない場合、またはデータ・タイプが間違っている場合に、例外が起こります。

パラメーター

item	検索される項目の名前。

戻り値 浮動値の入った配列。

例外

MQeException	Except_Type, ″wrong field type″
	Except_NotFound, item + ″ not found″

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  fields.restore( dumpData );
  ...
  float data[] = fields.getFloatArray( "Data" );
  ...
  }
```

MQeFields getInt

構文

```
public int getInt( String item ) throws Exception
```

説明 これは、MQeFields オブジェクトから int 型の長さの整数値項目を抽出します。データがない場合、またはデータ・タイプが間違っている場合に、例外が起こります。

パラメーター

item 検索される項目の名前。

戻り値 フィールドからの値の入った整数値。

例外

MQeException Except_Type, ″wrong field type″

Except_NotFound, item + ″ not found″

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  fields.restore( dumpData );
  ...
  int data = fields.getInt( "Data" );
  ...
  }
```

MQeFields getIntArray

構文

```
public int[] getIntArray( String item ) throws Exception
```

説明 これは、MQeFields オブジェクトから int 型の長さの整数値の動的配列を抽出し、それを配列として戻します。配列の長さは、この項目の *ArrayLength* 値によって決まります。データがない場合、またはデータ・タイプが間違っている場合に、例外が起こります。

パラメーター

item 検索される項目の名前。

戻り値 int 値の入った配列。

例外

MQeException Except_Type, ″wrong field type″

Except_NotFound, item + ″ not found″

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  fields.restore( dumpdDta );
  ...
  int data[] = fields.getIntArray( "Data" );
  ...
  }
```

MQeFields getLong

構文

```
public long getLong( String item ) throws Exception
```

説明 これは、MQeFields オブジェクトから long 型の長さの整数値項目を抽出します。データがない場合、またはデータ・タイプが間違っている場合に、例外が起こります。

パラメーター

item 検索される項目の名前。

戻り値 フィールドからの値の入った long 値。

例外

MQeException	Except_Type, ″wrong field type″
	Except_NotFound, item + ″ not found″

例

```
class MyApplication
   {
   ...
   MQeFields fields = new MQeFields( );
   fields.restore( dumpData );
   ...
   long data = fields.getLong( "Data" );
   ...
   }
```

MQeFields getLongArray

構文

```
public long[] getLongArray( String item ) throws Exception
```

説明 これは、MQeFields オブジェクトから long 型の長さの整数値の動的配列を抽出し、それを配列として戻します。配列の長さは、この項目の *ArrayLength* 値によって決まります。データがない場合、またはデータ・タイプが間違っている場合に、例外が起こります。

パラメーター

item 検索される項目の名前。

戻り値 long 値の入った配列。

例外

MQeException	Except_Type, ″wrong field type″
	Except_NotFound, item + ″ not found″

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  fields.restore( dumpData );
  ...
  long data[] = fields.getLongArray( "Data" );
  ...
  }
```

MQeFields getShort

構文

```
public short getShort( String item ) throws Exception
```

説明 これは、MQeFields オブジェクトから short 型の長さの整数値項目を抽出します。データがない場合、またはデータ・タイプが間違っている場合に、例外が起こります。

パラメーター

item 検索される項目の名前。

戻り値 フィールドからの値の入った short 型の整数。

例外

MQeException	Except_Type, ″wrong field type″
	Except_NotFound, item + ″ not found″

例

```
class MyApplication
   {
   ...
   MQeFields fields = new MQeFields( );
   fields.restore( dumpData );
   ...
   short data = fields.getShort( "Data" );
   ...
   }
```

MQeFields getShortArray

構文

```
public short[] getShortArray( String item ) throws Exception
```

説明 これは、MQeFields オブジェクトから short 型の長さの整数値の動的配列を抽出し、それを配列として戻します。配列の長さは、この項目の *ArrayLength* 値によって決まります。データがない場合、またはデータ・タイプが間違っている場合に、例外が起こります。

パラメーター

item 検索される項目の名前。

戻り値 short 値の入った配列。

例外

MQeException	Except_Type, ″wrong field type″
	Except_NotFound, item + ″ not found″

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  fields.restore( dumpData );
  ...
  short data[] = fields.getShortArray( "Data" );
  ...
  }
```

MQeFields getUnicode

構文

```
public String getUnicode( String item ) throws Exception
```

説明 これは、メッセージ・オブジェクトから Unicode データを抽出し、それをストリングとして戻します。データがない場合、またはデータ・タイプが間違っている場合に、例外が起こります。

パラメーター

item 検索される項目の名前。

戻り値 メッセージからの Unicode データが入ったストリング。

例外

MQeException	Except_Type, ″wrong field type″
	Except_NotFound, item + ″ not found″

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  fields.restore( dumpData );
  ...
  String data = fields.getUnicode( "Data" );
  ...
  }
```

MQeFields getUnicodeArray

構文

```
public String[] getUnicodeArray( String item ) throws Exception
```

説明 これは、メッセージ・オブジェクトから Unicode データを抽出し、それをストリングの配列として戻します。データがない場合、またはデータ・タイプが間違っている場合に、例外が起こります。

パラメーター

item 検索される項目の名前。

戻り値 メッセージからの Unicode データが入ったストリングの配列。

例外

MQeException Except_Type, ″wrong field type″

Except_NotFound, item + ″ not found″

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  fields.restore( dumpData );
  ...
  String data[] = fields.getUnicodeArray( "Data" );
  ...
  }
```

MQeFields hide

構文

```
public void hide( String item, boolean state ) throws Exception
```

説明 これは、MQeFields オブジェクトに対して等価性のテストを行うときに、MQeFields オブジェクト内の項目を組み込む (true) か、組み込まない (false) かを設定します。

パラメーター

item 隠される / 組み込まれる項目の名前。

state 隠す (true) または組み込む (false)

戻り値 なし

例外

MQeException Except_NotFound, item + ″ not found″

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( dumpData );
  ...
  fields.hide( "Data" );
  if ( OldFields.equals( fields ) )
    ...
  ...
  }
```

MQeFields putArrayLength

構文

```
public void putArrayLength( String item, int length ) throws Exception
```

説明 これは、指定項目の動的配列長さを設定します。

パラメーター

item 設定される項目の名前。

length エレメントの数の中の配列の長さ。

戻り値 なし

例外 なし

例

```
class MyApplication
 {
  ...
  MQeFields fields = new MQeFields( dumpData );
  ...
  fields.putArrayLength( "Data", 5 );
  ...
  }
```

MQeFields putArrayOfByte

構文

```
public void putArrayOfByte( String item,  byte data ) throws Exception
```

説明 このメソッドは、提供されたバイト配列のメッセージ・オブジェクトにデータを設定します。

パラメーター

item 設定される項目の名前。

data メッセージ・オブジェクトに設定されるデータの入ったバイト配列。

戻り値 なし

例外

MQeException さまざまなものがあります。

例

```
class MyApplication
 {
  ...
  MQeFields fields = new MQeFields( );
  ...
  fields.putArrayOfByte( "Data", new byte[] { 1, 2, 3, 4 } );
 ...
 }
```

MQeFields putArrayOfDouble

構文

```
public void putArrayOfDouble( String item, double data[] )
                                                    throws Exception
```

説明 これは、MQeFields オブジェクトに倍精度浮動小数点数の配列を設定します。

パラメーター

item 設定される項目の名前。

data MQeFields オブジェクトにコピーされる倍精度の値の配列。

戻り値 なし

例外 なし

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  fields.restore( dumpData );
  ...
 double data[] = fields.putArrayOfDouble( "Data" );
  ...
  }
```

MQeFields putArrayOfFloat

構文

```
public void putArrayOfFloat( String item, float data[] )
                                                    throws Exception
```

説明 これは、MQeFields オブジェクトに浮動小数点数の配列を設定します。

パラメーター

item 設定される項目の名前。

data MQeFields オブジェクトにコピーされる浮動値の配列。

戻り値 なし

例外 なし

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  fields.restore( dumpData );
  ...
  float data[] = fields.putArrayOfFloat( "Data" );
  ...
  }
```

MQeFields putArrayOfInt

構文

```
public void putArrayOfInt( String item, int data[] ) throws Exception
```

説明 これは、MQeFields オブジェクトに int 型の長さの整数値の配列を設定します。

パラメーター

item 設定される項目の名前。

data MQeFields オブジェクトにコピーされる int 値の配列。

戻り値 なし

例外 なし

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields();
  fields.restore( dumpData );
  ...
  fields.putArrayOfInt( "Data",new int[] { 1, 2, 3, 4 } );
  ...
  }
```

MQeFields putArrayOfLong

構文

```
public void putArrayOfLong( String item, long data[] ) throws Exception
```

説明 これは、MQeFields オブジェクトに long 型の長さの整数値の配列を設定します。

パラメーター

item 設定される項目の名前。

data MQeFields オブジェクトにコピーされる long 値の配列。

戻り値 なし

例外 なし

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields();
  fields.restore( dumpData );
  ...
  fields.putArrayOfLong( "Data",new long[] { 1, 2, 3, 4 } );
  ...
  }
```

MQeFields putArrayOfShort

構文

```
public void putArrayOfShort( String item, short data[] )
                                                  throws Exception
```

説明 これは、MQeFields オブジェクトに short 型の長さの整数値の配列を設定します。

パラメーター

item 検索される項目の名前。

data MQeFields オブジェクトにコピーされる short 値の配列。

戻り値 なし

例外 なし

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  fields.restore( dumpData );
```

```
...
fields.putArrayOfShort( "Data", new short[] { 1, 2, 3, 4 } );
...
}
```

MQeFields putAscii

構文

```
public void putAscii( String item, String data )  throws Exception
```

説明 このメソッドは、ASCII データ (108ページの『MQeFields getAscii』の注を参照) を MQeFields オブジェクトに設定し、データ・タイプを設定します。

パラメーター

item 設定される項目の名前。

data MQeFields オブジェクトに設定されるデータの入ったストリング。

戻り値 なし

例外 なし

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  ...
  fields.putAscii( "Data", "This is some data" );
  ...
  }
```

MQeFields putAsciiArray

構文

```
public void putAsciiArray( String item, String data[] ) throws Exception
```

説明 このメソッドは、ストリング配列から ASCII データ (108ページの『MQeFields getAscii』の注を参照) を MQeFields オブジェクトに設定し、データ・タイプを設定します。

パラメーター

item 設定される項目の名前。

data MQeFields オブジェクトに設定されるデータの入ったストリング配列。

戻り値 なし

例外

MQeException さまざまなものがあります。

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  ...
  String data[] = { "This is some data", "This is more data" };
  fields.putAsciiArray( "Data", data[] );
  ...
  }
```

MQeFields putBoolean

構文

```
public void putBoolean( String item, boolean data ) throws Exception
```

説明 これは、MQeFields オブジェクトにブール値を設定します。

パラメーター

item 設定される項目の名前。

data 設定されるブール値。

戻り値 なし

例外 なし

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  fields.restore( dumpData );
  ...
  fields.putBoolean( "Data", false );
  ...
  }
```

MQeFields putByte

構文

```
public void putByte( String item, byte data ) throws Exception
```

説明 このメソッドは、指定されたバイトのデータを MQeFields オブジェクトに設定します。

パラメーター

item 設定される項目の名前。

data メッセージ・オブジェクトに設定されるデータの入ったバイト。

戻り値 なし

例外 なし

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  ...
  fields.putByte( "Data", 123 );
  ...
  }
```

MQeFields putByteArray

構文

```
public void putByteArray( String item, byte data[][] ) throws Exception
```

説明 このメソッドは、バイト配列の指定された配列のデータを MQeFields オブジェクトに設定します。

パラメーター

item 設定される項目の名前。

data MQeFields オブジェクトに設定されるデータの入ったバイト配列の配列。

戻り値 なし

例外 なし

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  ...
  byte data[][] = new byte[2][];
  data[1][] = { 1, 2, 3, 4 };
  data[2][] = { 5, 6, 7, 8 };
  fields.putByteArray( "Data", data );
  ...
  }
```

MQeFields putDouble

構文

```
public void putDouble( String item, double data ) throws Exception
```

説明 このメソッドは、指定された倍精度の値のデータを MQeFields オブジェクトに設定します。

パラメーター

item 設定される項目の名前。

data MQeFields オブジェクトに設定される倍精度の値。

戻り値 なし

例外 なし

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  ...
  fields.putDouble( "Data", 123.456 );
  ...
  }
```

MQeFields putDoubleArray

構文

```
public void putDoubleArray( String item, byte data[][] )
                                                 throws Exception
```

説明 このメソッドは、2 倍長の浮動小数点値の配列を MQeFields に設定します。

パラメーター

item 設定される項目の名前。

data MQeFields オブジェクトに設定される倍精度の値の配列。

戻り値 なし

例外 なし

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  ...
  double data[] = new byte[2];
  data[1] = 1.234;
  data[2] = 5.678;
  fields.putDoubleArray( "Data", data );
  ...
  }
```

MQeFields putFields

構文

```
public void putFields( String item, MQeFields data ) throws Exception
```

説明 このメソッドは、この MQeFields オブジェクト内の項目として、データ・フィールドを設定します。

パラメーター

item 設定される項目の名前。

data この MQeFields オブジェクトに設定されるフィールド。

戻り値 なし

例外 なし

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  ...
  MQeFields subFields = new MQeFields( );
  ...
  fields.putFields( "Data", subFields );
  ...
  }
```

MQeFields putFieldsArray

構文

```
public void putFieldsArray( String item, MQeFields data[] )
                                                  throws Exception
```

説明 このメソッドは、この MQeFields オブジェクトにフィールドの配列を設定します。

パラメーター

item 設定される項目の名前。

data この MQeFields オブジェクトに設定されるフィールドの配列。

戻り値 なし

例外 なし

例

```
class MyApplication
  {
  ...
```

```
MQeFields fields = new MQeFields( );
...
MQeFields subFields = new MQeFields[2];
MQeFields subFields[0] = new MQeFields( );
MQeFields subFields[1] = new MQeFields( );
...
fields.putFieldsArray( "Data", subFields );
...
}
```

MQeFields putFloat

構文

```
public void putFloat( String item, float data ) throws Exception
```

説明 このメソッドは、指定された浮動値のデータを MQeFields オブジェクトに設定します。

パラメーター

item 設定される項目の名前。

data MQeFields オブジェクトに設定される浮動値。

戻り値 なし

例外 なし

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  ...
  fields.putFloat( "Data", 123.456 );
  ...
  }
```

MQeFields putFloatArray

構文

```
public void putFloatArray( String item, float data[] ) throws Exception
```

説明 このメソッドは、MQeFields オブジェクトに浮動小数点値の配列を設定します。

パラメーター

item 設定される項目の名前。

data MQeFields オブジェクトに設定される浮動小数点値の配列。

戻り値 なし

例外 なし

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  ...
  float data[] = new byte[2];
  data[1] = 1.234;   data[2] = 5.678;
  fields.putFloatArray( "Data", data );
  ...
  }
```

MQeFields putInt

構文

```
public void putInt( String item, int data ) throws Exception
```

説明 このメソッドは、MQeFields オブジェクトに整数を設定します。

パラメーター

item 設定される項目の名前。

data MQeFields オブジェクトに設定される整数値。

戻り値 なし

例外 なし

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  ...
  fields.putInt( "Data", 123456 );
  ...
  }
```

MQeFields putIntArray

構文

```
public void putIntArray( String item, int data[] ) throws Exception
```

説明 このメソッドは、MQeFields オブジェクトに整数値の配列を設定します。

パラメーター

item 設定される項目の名前。

data MQeFields オブジェクトに設定される整数値の配列。

戻り値 なし

例外 なし

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  ...
  int data[] = new byte[2];
  data[1] = 1234;   data[2] = 5678;
  fields.putIntArray( "Data", data );
  ...
  }
```

MQeFields putLong

構文

```
public void putLong( String item, long data ) throws Exception
```

説明 このメソッドは、MQeFields オブジェクトに long 値を設定します。

パラメーター

item 設定される項目の名前。

data MQeFields オブジェクトに設定される long 値。

戻り値

例外 なし

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  ...
  fields.putLong( "Data", 123456 );
  ...
  }
```

MQeFields putLongArray

構文

```
public void putLongArray( String item, long data[] ) throws Exception
```

説明 このメソッドは、MQeFields オブジェクトに long 値の配列を設定します。

パラメーター

item 設定される項目の名前。

data MQeFields オブジェクトに設定される long 値の配列。

戻り値 なし

例外 なし

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  ...
  long data[] = new byte[2];
  data[1] = 1234;   data[2] = 5678;
  fields.putLongArray( "Data", data );
  ...
  }
```

MQeFields putShort

構文

```
public void putShort( String item, short data ) throws Exception
```

説明 このメソッドは、MQeFields オブジェクトに short 値を設定します。

パラメーター

item 設定される項目の名前。

data MQeFields オブジェクトに設定される short 値。

戻り値 なし

例外 なし

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  ...
  fields.putShort( "Data", 123 );
  ...
  }
```

MQeFields putShortArray

構文

```
public void putShortArray( String item, short data[] ) throws Exception
```

説明 このメソッドは、MQeFields オブジェクトに short 値の配列を設定します。

パラメーター

item 設定される項目の名前。

data MQeFields オブジェクトに設定される short 値の配列。

戻り値 なし

例外 なし

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  ...
  short data[] = new byte[2];
  data[1] = 1234;
  data[2] = 5678;
  fields.putShortArray( "Data", data );
  ...
  }
```

MQeFields putUnicode

構文

```
public void putUnicode( String item, String data )   throws Exception
```

説明 このメソッドは、メッセージ・オブジェクト内に Unicode データを設定し、データ・タイプを設定します。

パラメーター

item 設定される項目の名前。

data メッセージ・オブジェクトに設定されるデータの入ったストリング。

戻り値 なし

例外 なし

例

```
class MyApplication
  {
  ...
  MQeFields msg = new MQeFields( );
  ...
  msg.putUnicode( "Data", "Merry xmas to all our readers" );
  ...
  }
```

MQeFields putUnicodeArray

構文

```
public void putUnicodeArray( String item, String data[] )
                                                  throws Exception
```

説明 このメソッドは、メッセージ・オブジェクト内にストリング配列のための Unicode データを設定し、データ・タイプを設定します。

パラメーター

item 設定される項目の名前。

data メッセージ・オブジェクトに設定されるデータの入ったストリングの配列。

戻り値 なし

例外 なし

例

```
class MyApplication
  {
  ...
  MQeFields msg = new MQeFields( );
  ...
  String data[] = new String[2];
  data[1] = "Merry xmas to all our readers";
  data[2] = "and a happy new year";
                        msg.putUnicode( "Data", data );
  ...
  }
```

MQeFields rename

構文

```
public void rename( String itemName, String newName ) throws Exception
```

説明 このメソッドは、MQeFields オブジェクト内の既存の項目を、指定された新しい名前に変更します。 *newName* を持つ項目がすでに MQeFields オブジェクトに存在する場合、それは、名前変更された項目で置き換えられます。

パラメーター

itemName 名前変更される項目の名前の入ったストリング。

newName 項目の新しい名前の入ったストリング。

戻り値 なし

例外

MQeException Except_NotFound, Item + ″ not found″

例

```
class MyApplication
  {
  ...
dumpDatafields.rename( "ThisItem", "ThatItem" );
  ...
  }
```

MQeFields restore

構文

```
public void restore( byte data[] ) throws Exception
```

説明 このメソッドは、 dump メソッドを使って作成されたバイト配列からメッセージ・オブジェクトを復元します。

パラメーター

data ダンプされた MQeFields オブジェクトの入ったバイト配列

戻り値 なし

例外

MQeException	Except_Type, ″wrong field type″
	Except_NotFound, item + ″ not found″
	Except_data, ″data:xxxx″
	Except_Type, ″Type: aaaa - bbbb″

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  ...
  fields.restore( dumpData );
  ...
  }
```

MQeFields restoreFromFile

構文

1.
```
public MQeFields restoreFromFile( String fileName,
                                  MQeAttribute attribute ) throws Exception
```

2.
```
public MQeFields restoreFromFile( String fileName,
                                  String endRecord,
                                  String sectionMatch,
                                  String template ) throws Exception
```

説明 これらのメソッドは、ディスク・ファイルの内容から新しい MQeFields オブジェクトを作成します。 2 つの形式があります。

1. ダンプされた MQeFields オブジェクトからのバイト配列の入ったバイナリー・ファイル

2. レコードによって区切られたセクションの入った ASCII ファイル。さらにレコードは、フィールド内で、*endRecord* ストリングとネストされたフィールドによって *sectionMatch* および *endRecord* 区切りストリングから分けられています。

パラメーター

fileName 読み取られるファイルの名前が入ったストリング。

attribute バイナリー・データをデコード (暗号化解除および (または) 圧縮解除) するために使用される MQeAdminQueueAdminMsg オブジェクト。

endRecord レコード終わりを区切る文字の入ったストリング。たとえば、¥r¥n

sectionMatch 以下のためのパターン・テンプレートの入ったストリング。

- セクション名。たとえば、[#0]
- レコード終わりを区切る文字。たとえば、¥r¥n

sectionMatch テンプレートの挿入シーケンスは #0 だけです。

template 入力の構文解析に使用するストリング・テンプレート。

テンプレートには、以下のように、最大で 3 つの挿入シーケンス #n があります。

#0 データ・タイプ用

#1 フィールド名用

#2 フィールド値用

以下に例を示します。

Name=#1, Type=#0, Value=#2

戻り値 復元される値の入った MQeFields オブジェクト。

例外 さまざまな変換の例外があります。

例

```
class MyApplication
  {
  ...
  MQeFields fields = MQeFields.restoreFromFile(File.separator + "directory" +
                                               File.separator + "thisfile.xyz",
                                               "¥r¥n",
                                               "[#0]",
                                               "#1=#2" );
  ...
  }
```

上記の例では、以下の構造を持つ ASCII ファイルを処理しています。

```
[Section1]
item1=1235678
item2=abcdef
[Section2]
item1=qwertyiop
```

これは、項目名が *Section1* と *Section2* の 2 つの組み込みフィールド・オブジェクトの入った MQeFields オブジェクトを構成します。これらの各組み込みフィールドには該当するセクションからの関連項目が入っています。

MQeFields restoreFromString

構文

1.
```
public void restoreFromString( String template,
                               String data   ) throws Exception
```
2.
```
public void restoreFromString( String endRecord,
                               String template,
                               String data   ) throws Exception
```
3.
```
public static MQeFields restoreFromString( String endRecord,
                                           String sectionMatch,
                                           String template,
                                           String data   ) throws Exception
```

説明 これらのメソッドは以下を復元します。

1. ストリングからの個々の項目。
2. *EndRecord* 区切りストリングからの項目のグループ。
3. *SectionMatch* および *EndRecord* 区切りストリングからの、 MQeFields オブジェクト内でネストされている MQeFields。

パラメーター

template

入力の構文解析に使用するストリング・テンプレート。

テンプレートには、以下のように、最大で 3 つの挿入シーケンス #n があります。

#0 データ・タイプ用

#1 フィールド名用

#2 フィールド値用

以下に例を示します。

```
Name=#1, Type=#0, Value=#2
```

戻り値 復元される値の入った MQeFields オブジェクト。

例外 さまざまな変換の例外があります。

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  fields.putBoolean( "tb", true );
  ...
  fields.putLong( "ml", -1 );
  String data = fields.dumpToString( "Name=#1, Type=#0 Value=#2¥r¥n" ) );
  ...
  MQeFields newFields = new MQeFields( );
  newFields.restoreFromString( "Name=#1, Type=#0 Value=#2¥r¥n", data );
   ...
  }
```

MQeFields setAttribute

構文

```
public void setAttribute( MQeAttribute attribute ) throws Exception
```

説明 このメソッドは、ダンプまたは復元する際に必ず、MQeFields オブジェクトの内容をエンコードまたはデコードするために使用される属性を割り当てます。

パラメーター

attribute MQeAdminQueueAdminMsg オブジェクト参照

戻り値 なし

例外 なし

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  ...
  MQeAttribute attr = new MQeAttribute( null, new MQeXorCryptor( ), null );
  fields.setAttribute( attr );
  ...
  }
```

MQeFields updateValue

構文

```
public long updateValue( String item, long update ) throws Exception
```

説明 このメソッドは、この MQeFields オブジェクト内に保持されている整数値を増分または減分します。

パラメーター

item 設定される項目の名前。

update 指定された項目の現在の値に追加される値。

戻り値 更新された値。

例外

MQeException さまざまなものがあります。

例

```
class MyApplication
  {
  ...
  MQeFields fields = new MQeFields( );
  ...
  fields.putInt( "Data", 123 );
  ...
  long l = fields.updateValue("Data", -3 );
  ...
  }
```

MQeKey

このクラスは、MQeKey オブジェクトを作成するために使用します。 MQeKey オブジェクトは、属性オブジェクトに付加され、属性オブジェクトで使用することができます。属性は、チャネルおよび MQeFields オブジェクトに関連付けられています。

パッケージ **com.ibm.mqe**

このクラスは MQe の下位クラスです。

コンストラクター

MQeKey

構文

```
public MQeKey( )
```

記述 MQeKey オブジェクトを作成する

パラメーター

なし

戻り値 なし

例外 なし

関連する関数

- MQeAttribute
- MQeMAttribute

メソッド

メソッド	目的
setLocalKey	ローカルの暗号化キーと暗号化解除キーを、提供された暗号キー・シード値から派生した値に設定する

MQeKey setLocalKey

構文

```
Public void setLocalKey ( String localCipherKey ) throws MQeException
```

記述 データを保護し、特定のターゲット・ファイル名に書き込む

パラメーター

localCipherKey キーの暗号化キーと暗号化解除キーの派生元になるシード値

戻り値 なし

例外

MQeException	Except_NotAllowed, ″invalid localCipherKey″

例

```
class MySampleClass extends MQe
  {
  try
    {
    /* protecting MQeFields data                          */
    MQeDESCryptor des  = new MQeDESCryptor( );
    MQeAttribute desA  = new MQeAttribute( null, des, null);
    MQeKey localkey    = new MQeKey();
    localkey.setLocalKey( "It_is_a_secret");
    desA.setKey( localkey );
    MQeFields localf   = new MQeFields( );
    localf.setAttribute( desA );
    Trace ( "i: test data = " + "0123456789abcdef...." );
    localf.putArrayOfByte(
          "TestData", asciiToByte("0123456789abcdef....") );
    byte[] temp        = localf.dump( );
    Trace ( "i: test data protected using MQeKey = " +
                                       byteToHex( temp ) );
    /* unprotecting MQeFields data                        */
    MQeDESCryptor des2 = new MQeDESCryptor( );
    MQeAttribute desA2 = new MQeAttribute( null, des2, null);
    MQeKey localkey2   = new MQeKey();
    localkey2.setLocalKey( "It_is_a_secret");
    desA2.setKey( localkey );
    MQeFields localf2   = new MQeFields( );
    localf2.setAttribute( desA2 );
    localf2.restore ( temp );
    Trace ( "i: unprotected test data = " +
          byteToAscii(localf2.getArrayOfByte( "TestData") );
    }
  catch ( Exception e )
    {
    }
```

関連する関数

- MQeLocalSecure

MQeMessageEvent

このオブジェクトは、MQSeries Everyplace メッセージ・イベントの発生時にアプリケーションに渡されます。

パッケージ **com.ibm.mqe**

java.util.EventObject を拡張します。

メソッド

メソッド	目的
getMsgFields	イベントを生成させたメッセージから選択されたフィールドを含む MQeFields オブジェクトを戻します。
getQueueManagerName	このイベントを生成したキューを所有するキュー・マネージャーの名前を含む String を戻します。
getQueueName	このイベントを生成したキューの名前を含む String を戻します。

MQeMessageEvent getMsgFields

構文

```
public MQeFields getMsgFields()
```

記述 このメソッドは、イベントを生成させたメッセージから選択されたフィールドを含む MQeFields オブジェクトを戻します。メッセージの *UID* (タイム・スタンプと起点キュー・マネージャー名で構成される) が常に戻されます。メッセージ ID、相関 ID、およびメッセージ優先順位値がメッセージ内に存在する場合は、それらも戻されます。

パラメーター

なし。

戻り値 イベントを生成させたメッセージから選択されたフィールドを含む MQeFields オブジェクト。

例外 なし。

例

```
class MyMQeApplication
{
  ...
  /* called when a msg event occurs */
  public void messageArrived( MQeMessageEvent e )
  {
    String eventQueueName = e.getQueueName(); /* get origin Q name */
    if ( eventQueueName.equals( "SYSTEM.DEFAULT.LOCAL.QUEUE" ) )
    {
      ...
      /* get msg info  */
      MQeFields filter = e.getMsgFields();
      System.out.println( "Message received from QueueMgr: " +
```

```
            e.getQueueManagerName() );
          qmgr.getMessage( null, "SYSTEM.DEFAULT.LOCAL.QUEUE", filter, null, 0 );
          ...
        }
        ...
      }
      ...
    }
```

MQeMessageEvent getQueueManagerName

構文

```
public String getQueueManagerName()
```

記述 このメソッドは、このイベントを生成したキューを所有するキュー・マネージャーの名前を含む String を戻します。

パラメーター

なし

戻り値 このイベントを生成したキューを所有するキュー・マネージャーの名前を含む String。

例外 なし

例

```
class MyMQeApplication
{
  ...
  /* called when a msg event occurs */
  public void messageArrived( MQeMessageEvent e )
  {
    String eventQMgr = e.getQueueManagerName(); /* get origin QMgr  */
    String eventQueueName = e.getQueueName(); /* get origin Q name */

    if ( eventQMgr.equals( localQMgr.getName() ) )
    { /* local QMgr */
      ...
    }
    ...
  }
  ...
}
```

関連する関数

getQueueName

MQeMessageEvent getQueueName

構文

```
public String getQueueName()
```

記述 このメソッドは、このイベントを生成したキューの名前を含む String を戻します。

パラメーター

なし

戻り値 このイベントを生成したキューの名前を含む String。

例外 なし

例

```
class MyMQeApplication
{
  ...
  /* called when a msg event occurs */
  public void messageArrived( MQeMessageEvent e )
  {
    String eventQueueName = e.getQueueName(); /* get origin Q name */
    if ( eventQueueName.equals( "SYSTEM.DEFAULT.LOCAL.QUEUE" ) )
    {
      ...
    }
    ...
    if ( eventQueueName.equals( "MyQueue" ) )
    {
      ...
    }
  }
  ...
}
```

関連する関数

getQueueManagerName

MQeMsgObject

この節では、基本の MQeMsgObject を作成するために使用する Java クラスについて説明します。このオブジェクトを使用して、データを保持したり、1 つの MQSeries Everyplace システムから別の MQSeries Everyplace システムに送信するデータを取得するために必要な論理を組み込んだりします。通常、追加の特性、データまたはコードを保持するには、このクラスの下位クラスが使用されます。

パッケージ　　com.ibm.MQe

このクラスは MQeFields の下位クラスです。

定数と変数

MQeMsgObject では、MQSeries Everyplace 基底クラスの以下のフィールド名定数が使用されます。

以下の 2 つの定数は、結合されて固有のメッセージ ID を構成します。これらの定数は MQSeries Everyplace システムによって設定され、アプリケーションではこれらの値を変更しようとしてはなりません。 *Msg_OriginQMgr* は ASCII 項目で、*Msg_Time* は long 整数値です。

```
public  final static String   Msg_OriginQMgr
public  final static String   Msg_Time
```

以下のフィールド名定数は、 MQSeries システムとの間でやり取りされるメッセージで使用するために用意されているもので、完全に MQSeries Everyplace 環境内で使用されるメッセージには必要ありません。

これらの定数は、バイト配列として処理しなければならず、 MQSeries に送られる場合は長さが 24 バイトでなければなりません。

```
public  final static String   Msg_CorrelID
public  final static String   Msg_MsgID
```

以下のフィールド名定数は、 MQSeries システムとの間でやり取りされるメッセージで使用するために用意されているもので、完全に MQSeries Everyplace 環境内で使用されるメッセージには必要ありません。これは、メッセージ・スタイルを設定するために使用されます。

```
public  final static String   Msg_Style
```

この定数は int 整数で、以下の値を取ります。

```
    public  final static int      Msg_Style_Datagram
    public  final static int      Msg_Style_Request
    public  final static int      Msg_Style_Reply
```

以下のフィールド名定数は、 MQSeries システムとの間でやり取りされるメッセージで使用するために用意されているもので、完全に MQSeries Everyplace 環境内で使用され

るメッセージには必要ありません。これらの定数は、 MQSeries Everyplace 環境でキューからメッセージを検索するアプリケーションで同じ意味を持たせることができます。

これらのフィールドはいずれも ASCII です。

```
public  final static String   Msg_ReplyToQ
public  final static String   Msg_ReplyToQMgr
```

以下のフィールド名定数は 0 ～ 9 の間のバイト値でなければならず、メッセージ優先順位を設定します。キューへの追加時にこの定数が設定されていないと、キューのデフォルト優先順位値が有効になります。

```
public  final static String   Msg_Priority
```

以下のフィールド名定数は、メッセージの有効期限を終わらせるために使用できます。MQSeries Everyplace は、有効期限が切れた場合にメッセージを廃棄できます。

この定数は、以下の 2 つのうちいずれかの意味を持ちます。

1. long 整数の場合、有効期限はメッセージを廃棄できるようになるまでの絶対日時と見なされます。
2. int 整数の場合、有効期限はメッセージ・オブジェクトの作成時刻を基準とした相対日時となります。

```
public  final static String   Msg_ExpireTime
```

以下のフィールド名は、アプリケーションまたはシステムが設定できる Boolean 値であり、メッセージが再送信されている (または、再送信された) ことを示します。

この再送フラグは、内部フローでの保証されたメッセージ送達を制御するために、MQSeries Everyplace システムによって設定またはリセットされます。

```
public  final static String   Msg_Resend
```

以下のフィールド名定数は、long 整数値であり、通常は、ロック状態でブラウズしている際に **browseMessages()** 要求によって戻される値です。この定数は、完全に MQSeries Everyplace 環境で使用されるメッセージには必要ありません。

```
public  final static String   Msg_LockID
```

コンストラクター

MQeMsgObject

構文

1. `public MQeMsgObject( )`
2. `public MQeMsgObject ( byte data[] )`
3. `public MQeMsgObject ( MQeMsgObject msg )`

記述 コンストラクターは、MQeMsgObject オブジェクトを作成して初期化します。このコンストラクターには次の 2 つの形式があります。

1. パラメーターなし。この場合、空のメッセージ・オブジェクトを構成します。
2. バイト配列付き。この場合、提供されたバイト配列からフィールド・オブジェクトを復元します。

   注: 各オブジェクトは同じタイプでなければなりません。
3. MQeMsgObject 付き。この場合、提供されたメッセージを新しいメッセージ・オブジェクトにラップします。データを強制的にエンコードさせる属性が付加されたラッパー・メッセージでは、通常これが使用されます。

パラメーター

data ダンプされたフィールド・オブジェクトを含むバイト配列。

戻り値 なし

例外

MQeException

Except_Type, ″wrong field type″

Except_NotFound, item + ″ not found″

Except_data, ″data:xxxx″

Except_Type, ″Type: aaaa - bbbb″

例

```
class MyApplication
  {
  ...
  MQeMsgObject msg = new MsgObject( );
  ...
  ...
  }
```

メソッドの要約

メソッド	目的
getMsgUIDFields	メッセージの固有 ID を抽出します。
getOriginQMgr	起点キュー・マネージャー (存在する場合) の名前を抽出します。
getTimeStamp	メッセージ・オブジェクトが作成された時刻を抽出します。
putOriginQMgr	メッセージの起点キュー・マネージャー名を設定します。 **注:** いったん設定すると、これは変更できません。
resetMsgUIDFields	メッセージ・オブジェクトの固有 ID をリセットします。
unwrapMessageObject	ラップされたメッセージ・オブジェクトをアンラップします。

MQeMsgObject getMsgUIDFields

構文

```
public MQeFields getMsgUIDFields ( )
```

記述 このメソッドは、以下のフィールド項目を含む MQeFields オブジェクトを戻します。

msg_Time	メッセージが作成された時刻
msg_OriginQMgr	起点キュー・マネージャーの名前

戻されたフィールド・オブジェクトは、等価性テスト、またはメッセージのブラウズまたは取得呼び出しで、一致パラメーターとして使用できます。

パラメーター

なし

戻り値 固有のメッセージ ID を作成するために使用されたフィールドを含む MQeFields オブジェクト。

例外

MQeException	Except_NotAllowed,'Queue Manager not set'

例

```
class MyApplication
{
...
...
MQeFields uid = Msg.getMsgIDFields( );
...
}
```

MQeMsgObject getOriginQMgr

構文

```
public String getOriginQMgr( )
```

記述 起点キュー・マネージャーの名前を含む String、またはヌル (設定されていない場合) を戻します。

パラメーター

なし

戻り値 String またはヌル

例外 なし

例

```
class MyApplication
{
...
```

```
...
String uid = Msg.getOriginQMgr();
...
}
```

MQeMsgObject getTimeStamp

構文

```
public long getTimeStamp( )
```

記述 オブジェクトが作成された時刻 (ミリ秒単位) を含む long 整数値を戻します。

パラメーター

なし

戻り値 ミリ秒単位の long 値

例外 なし

例

```
class MyApplication
{
...
...
long uid = Msg.getTimeStamp ( );
...
}
```

MQeMsgObject putOriginQMgr

構文

```
public void putOriginQMgr( )
```

記述 起点キュー・マネージャーの名前を設定します。いったん設定されると、この名前はリセットできません。

注: 通常、このメソッドは、**putMessage()** 呼び出しが発行されたときに、キュー・マネージャーによって内部で呼び出されるだけです。

パラメーター

なし

戻り値 なし

例外 なし

MQeMsgObject resetMsgUIDFields

構文

```
public void resetMsgUIDFields ( )
```

記述 このメソッドは、新しい *Msg_Time* 値が生成され、*Msg_OriginQMgr* がヌルに設

定されるようにメッセージ・オブジェクト *UID* をリセットします。その結果、新しいメッセージ・オブジェクトが作成されますが、設定されていたフィールド項目は保持されます。

パラメーター

なし

戻り値 なし

例外 なし

例

```
{
...
...
msg.resetMsgUIDFields ( );
...
}
```

MQeMsgObject unwrapMessageObject

構文

```
public MQeMsgObject unwrapMessageObject( MQeAttribute attribute )
```

記述 このメソッドは、組み込まれた MQeMsgObject をアンラップして、指定された属性を使ってデコードし (適宜)、新しいメッセージ・オブジェクトを戻します。

パラメーター

attribute MQeAttribute オブジェクト参照またはヌル。組み込まれたメッセージ・オブジェクトをデコードするために使用されます。

戻り値 なし

例外 なし

例

```
class MyApplication
  {
  ...
  ...
  msg.resetMsgUIDFields( );
  ...
  }
```

MQeQueue

MQeQueue は基底キュー・クラスで、他のすべてのタイプのキューはこのオブジェクトの下位クラスです。

キューとは、メッセージを保持するオブジェクトのことです。メッセージは、キューが所有するメッセージ・ストアに保持されます。通常、このメッセージ・ストアは、ハード・ディスクなどの永続的なストレージ・デバイスです。ただし、データベースなどの他のタイプのストアを使用することもできます。 MQSeries Everyplace は、キューに永続的なストアがあるという事実を信頼して、その保証されたメッセージ送達を提供することができます。したがって、何らかの永続的でないストレージを使用すると、MQSeries Everyplace の保証されたメッセージ送達は無効になります。

キューは、キュー・ストア・アダプター を使用して、ストレージ・デバイスとの通信を処理します。アダプターとは、 MQSeries Everyplace とハードウェア・デバイス (ディスクやネットワークなど) またはソフトウェア (データベースなど) の間のインターフェースのことです。アダプターはプラグ可能コンポーネントとして設計されているため、キュー・ストアは容易に変更することができます。

キューに保持されているメッセージは、オーセンティケーターと暗号機能によって保護できます。メッセージはまた、圧縮機能によって圧縮することもできます。オーセンティケーター、暗号機能、および圧縮機能は、キューの属性であり、キューに関連付ける適切な MQeAttribute オブジェクトを指定することによって定義されます。

キューの動作は一連の規則によって制御されます。これらの規則は Java クラスの形式を取り、 MQSeries Everyplace ソリューションによって拡張できます。キュー規則の基底セットは、 MQeQueueRule クラスで定義されています。キューの動作中、これらの規則は特定のイベント (たとえば、メッセージがプットされた、メッセージの有効期限が切れた、または重複メッセージが到着した) が発生したときに呼び出されます。次いで、これらの規則はキューがこれらのイベントを処理する方法を決定します。

キュー有効期限間隔を定義できます。キューの有効期限間隔を過ぎてもキューに残っているメッセージは、有効期限切れとしてマークされます。次いで、キューの規則により、メッセージに生じる事柄が決定されますが、通常は、メッセージが削除されるか、「送達不能キュー」に入れられます。キューの有効期限間隔は、メッセージの有効期限間隔とは異なります。

メッセージの最大数や、個々のメッセージの最大可能サイズも定義できます。

すべてのキューはキュー・マネージャーによって所有されます。キュー・マネージャーによって所有されるキューは、キュー・マネージャーの*ローカル・キュー* として認識されます。キュー・マネージャーは、他のキュー・マネージャーに属するキューにもアクセスできます。これらのキューは、*リモート・キュー* として認識されます。リモート・キューにアクセスする際、キュー・マネージャーはそのキューの特性について認識した

情報を保管します。その情報は、リモート・キュー定義に保管されます。リモート・キュー定義は MQeRemoteQueue クラスによって表されます。

パッケージ

com.ibm.mqe

メソッド

メソッド	目的
getCreationDate	このメソッドは、キューが作成された日時を表す long 値を戻します。
getDefaultPriority	このメソッドは、キューのデフォルト優先順位である整数値を戻します。
getDescription	このメソッドは、キューの記述ストリングを含む String オブジェクトを戻します。
getExpiryInterval	このメソッドは、キューのメッセージ有効期限間隔 (ミリ秒単位) である long 値を戻します。
getMaxMessageSize	このメソッドは、キューが保持できるメッセージの最大サイズ (バイト単位) である整数値を戻します。
·getMaxQueueSize	このメソッドは、このキューで保持できるメッセージの最大数である整数値を戻します。
getNumberOfMessages	このメソッドは、このキューで保持されている現在のメッセージ数である整数値を戻します。
getQueueAttribute	このメソッドは MQeAttribute オブジェクトを戻します。これは、このキューが使用するオーセンティケーター、暗号機能、および圧縮機能を定義します。
getQueueName	このメソッドは、キューの名前を含む String オブジェクトを戻します。
getQueueStore	このメソッドは、キューの永続的なストアのパス名を含む String を戻します。

MQeQueue getCreationDate

構文

```
public long getCreationDate()
```

記述 このメソッドは、このキューが作成された日時を戻します。

パラメーター

なし

戻り値 このキューが作成された時刻を表す long 値。

例外 なし

例

```
class ExampleQueueManagerRules extends MQeQueueManagerRule
{
  ...
  public void addQueue( MQeQueue queue ) throws MQeException
  {
    /* only allow the addition of queues created after 1st June 2000 */
    Calendar calendar = Calendar.getInstance();
    calendar.set( 2000, 05, 01 );
    Date date = calendar.getTime();
    /* get current time */
    Date qDate = new Date( queue.getCreationDate() );
    /* compare the two dates */
    if ( date.after( qDate ) )
      throw new MQeException( Except_Rule, "addQueue disallowed" );
  }
  ...
}
```

MQeQueue getDefaultPriority

構文

```
public int getDefaultPriority()
```

記述 このメソッドは、キューのデフォルト優先順位値を戻します。この優先順位値は、キューに入れられたメッセージのうち、以前に優先順位値を割り当てられていないものに使用されます。

パラメーター

なし

戻り値 このキューのデフォルト優先順位である整数値。

例外

MQeException	なし

例

```
class ExampleQueueManagerRules extends MQeQueueManagerRule
{
  ...
  public boolean transmit( MQeQueue queue )
  {
    /* allow transmission if queue's priority is greater than 5 */
    if ( queue.getDefaultPriority() > 5 )
      return (true);
    else
      return (false);
  }
  ...
}
```

MQeQueue getDescription

構文

```
public String getDescription()
```

記述 このメソッドは、このキューの記述ストリングを戻します。

パラメーター

なし

戻り値 このキューの記述ストリングを含む String オブジェクト。

例外 なし

例

```
import examples.eventlog.*;

class ExampleQueueManagerRules extends MQeQueueManagerRule
{
  public final static int Event_QueueManager_AddQueue = 201;
  LogToDiskFile logFile = null;
  ...
  public void addQueue( MQeQueue queue )
  {
    /* log the addition of a new queue */
    log( MQe_Log_Information, Event_QueueManager_AddQueue,
         "Queue " + queue.getQueueManagerName() + "+" + queue.getQueueName() +
         ": " + queue.getDescription() );
  }

  public void queueManagerActivate() throws Exception
  {
    /* create a new log file */
    logFile = new LogToDiskFile( "¥¥log.txt");
  }

  public void queueManagerClose()
  {
    /* close log file */
    logFile.close();
  }
  ...
}
```

MQeQueue getExpiryInterval

構文

```
public long getExpiryInterval()
```

記述 このメソッドは、このキューのメッセージ有効期限間隔を戻します。キューの有効期限間隔を過ぎてもキューに残っているメッセージは、有効期限切れとしてマークされます。次いで、キューの規則により、メッセージに生じる事柄が決定されます。

パラメーター

なし

戻り値 このキューのメッセージ有効期限間隔 (ミリ秒単位) である long 値。値 0 は、キューにメッセージ有効期限間隔が設定されていないことを意味します。

例外 なし

例

```
class ExampleQueueManagerRules extends MQeQueueManagerRule
{
  ...
  public boolean transmit( MQeQueue queue )
  {
    /* transmit if queue has a low message expiry time (less than 1 day) */
    /* (zero means no expiry) */
    if ( queue.getExpiryInterval() < (60 * 60 * 24 * 1000) &&
         queue.getExpiryInterval() > 0 )
      return (true);
    else
      return (false);
  }
  ...
}
```

MQeQueue getMaxMessageSize

構文

```
public int getMaxMessageSize()
```

記述 このメソッドは、このキューで保持できるメッセージの最大サイズ (バイト単位) を戻します。

パラメーター
なし

戻り値 このキューで保持できるメッセージの最大サイズ (バイト単位) である整数値。

例外 なし

例

```
{
  ...
  public void addQueue( MQeQueue queue ) throws MQeException
  {
    /* only allow addition of queue if it supports messages of at least 2MB */
    if ( queue.getMaxMessageSize() < 2048000 )
      throw new MQeException( Except_Rule, "Message size too small" );
  }
  ...
}
```

MQeQueue getMaxQueueSize

構文

```
public int getMaxQueueSize()
```

記述 このメソッドは、このキューで保持できるメッセージの最大数を戻します。

パラメーター

なし

戻り値 このキューで保持できるメッセージの最大数である整数値。

例外 なし

例

```
class ExampleQueueManagerRules extends MQeQueueManagerRule
{
  ...
  public void addQueue( MQeQueue queue ) throws MQeException
  {
    /* only allow addition of queue if it supports more than 100 messages */
    if ( queue.getMaxQueueSize() < 100 )
      throw new MQeException( Except_Rule, "Max Queue depth too small" );
  }
  ...
}
```

MQeQueue getNumberOfMessages

構文

```
public int getNumberOfMessages()
```

記述 このメソッドは、このキューで保持されている現在のメッセージ数を戻します。

パラメーター

なし

戻り値 このキューで保持されている現在のメッセージ数である整数値。

例外 なし

例

```
class ExampleQueueManagerRules extends MQeQueueManagerRule
{
  ...
  public boolean transmit( MQeQueue queue )
  {
    /* only allow queue to transmit if it contains more than 10 messages */
    if ( queue.getNumberOfMessages() >= 10 )
      return (true);
    else
      return (false);
  }
  ...
}
```

MQeQueue getQueueAttribute

構文

```
public MQeAttribute getQueueAttribute()
```

記述 このメソッドは、このキューの属性オブジェクトを戻します。属性オブジェク

トは、このキューが使用するオーセンティケーター、暗号機能、および圧縮機能を定義します。これらの属性は、キューに保管されているすべてのメッセージに使用されます。

パラメーター

なし

戻り値 キューが使用するオーセンティケーター、暗号機能、および圧縮機能を定義する MQeAdminQueueAdminMsg オブジェクト。

例外 なし

例

```
class ExampleQueueManagerRules extends MQeQueueManagerRule
{
  ...
  public void addQueue( MQeQueue queue ) throws Exception
  {
    /* only allow addition of queues with a defined DES Cryptor */
    MQeAttribute qAttribute = queue.getQueueAttribute();
    if ( qAttribute == null )
      throw new MQeException( Except_Rule, "No queue attribute defined" );

    MQeCryptor cryptor = qAttribute.getCryptor();
    if ( cryptor == null )
      throw new MQeException( Except_Rule, "No cryptor defined" );

    if ( !(cryptor.securityLevel().equals( "DES" )) )
      throw new MQeException( Except_Rule, "DES Cryptor not defined" );
  }
  ...
}
```

MQeQueue getQueueManagerName

構文

```
public String getQueueManagerName()
```

記述 このメソッドは、このキューが属するキュー・マネージャーの名前を戻します。

パラメーター

なし

戻り値 このキューが属するキュー・マネージャーの名前を含む String オブジェクト。

例外 なし

例

```
import examples.eventlog.*;

class ExampleQueueManagerRules extends MQeQueueManagerRule
{
  public final static int Event_QueueManager_AddQueue = 201;
  LogToDiskFile logFile = null;
  ...
  public void addQueue( MQeQueue queue )
```

```
  {
    /* log the addition of a new queue */
    log( MQe_Log_Information, Event_QueueManager_AddQueue,
         "Queue " + queue.getQueueManagerName() + "+" + queue.getQueueName() +
         ": " + queue.getDescription() );
  }

  public void queueManagerActivate() throws Exception
  {
    /* create a new log file */
    logFile = new LogToDiskFile( "¥¥log.txt");
  }

  public void queueManagerClose()
  {
    /* close log file */
    logFile.close();
  }
  ...
}
```

関連する関数

getQueueName

MQeQueue getQueueName

構文

```
public String getQueueName()
```

記述 このメソッドは、このキューの名前を戻します。

パラメーター

なし

戻り値 このキューの名前である String オブジェクト。

例外 なし

例

```
import examples.eventlog.*;

class ExampleQueueManagerRules extends MQeQueueManagerRule
{
  public final static int Event_QueueManager_AddQueue = 201;
  LogToDiskFile logFile = null;
  ...
  public void addQueue( MQeQueue queue )
  {
    /* log the addition of a new queue */
    log( MQe_Log_Information, Event_QueueManager_AddQueue,
         "Queue " + queue.getQueueManagerName() + "+" + queue.getQueueName() +
         ": " + queue.getDescription() );
  }

  public void queueManagerActivate() throws Exception
  {
    /* create a new log file */
    logFile = new LogToDiskFile( "¥¥log.txt");
  }

  public void queueManagerClose()
  {
```

```
    /* close log file */
    logFile.close();
  }
  ...
}
```

関連する関数

getQueueManagerName

MQeQueue getQueueStore

構文

```
public String getQueueStore()
```

記述 このメソッドは、キューの永続的なストアへのパス名を戻します。

パラメーター

なし

戻り値 キューの永続的なストアへのパス名である String オブジェクト。

例外 なし

例

```
import examples.eventlog.*;

class ExampleQueueManagerRules extends MQeQueueManagerRule
{
  public final static int Event_QueueManager_AddQueue = 201;
  LogToDiskFile logFile = null;
  ...
  public void addQueue( MQeQueue queue )
  {
    /* log the addition of a new queue */
    log( MQe_Log_Information, Event_QueueManager_AddQueue,
         "Queue " + queue.getQueueManagerName() + "+" + queue.getQueueName() +
         ": " + queue.getDescription() + ": Persistent store: " +
         queue.getQueueStore() );
  }

  public void queueManagerActivate() throws Exception
  {
    /* create a new log file */
    logFile = new LogToDiskFile( "¥¥log.txt");
  }

  public void queueManagerClose()
  {
    /* close log file */
    logFile.close();
  }
  ...
}
```

MQeQueueManager

このクラスは、MQSeries Everyplace キュー・マネージャー・オブジェクトを構成するために使用します。

MQSeries Everyplace キュー・マネージャーは、MQSeries Everyplace システムのフォーカル・ポイントです。これは、以下のものを提供します。

- MQSeries Everyplace アプリケーションが使用する MQSeries Everyplace および MQSeries ネットワークへのアクセスの中心点
- 一度限りの保証されたメッセージ送達
- 障害条件からの完全なリカバリー
- 拡張可能なルールに基づく動作

パッケージ

com.ibm.mqe

コンストラクター

MQeQueueManager

構文

```
public MQeQueueManager( )
```

記述 MQeQueueManager オブジェクトを構成します。**activate()** メソッドを呼び出して、キュー・マネージャーを初期化して開始する必要があります。

注: キュー・マネージャーを開始する前に、必要なすべての MQSeries Everyplace 別名を追加する必要があります。これを行う方法について、詳しくは、「*MQSeries Everyplace for Multiplatforms プログラミング・ガイド*」を参照してください。

パラメーター

なし

戻り値 なし

例外

MQeException	`Except_QMgr_Activated`
	`Except_QMgr_AlreadyExists`
	`Except_QMgr_InvalidQMgrName`
	`Except_QMgr_NotConfigured`

例

```
import com.ibm.mqe.*;
public class StartQueueManager {
public static void main(String[] args) {
try {
if (null == args || args.length < 1 || null == args[0]) {
System.out.println("No ini file name supplied");
} else {
System.out.println("Reading ini file: " + args[0]);
MQeFields iniSections =  MQeFields.restoreFromFile(args[0],
                                                   "\r\n", "[#0]",
                                                   "(#0)#1=#2");
MQeQueueManager queueManager = new MQeQueueManager();
System.out.println("Activating queueManager");
queueManager.activate(iniSections);
System.out.println("Closing queueManager");
queueManager.close();
System.out.println("Closed queueManager");
}
} catch (Exception e) {
System.out.println("Yet another exception thrown: " + e.getMessage());
e.printStackTrace();
}
}
}
```

メソッド

メソッド	目的
activate	すでにインスタンス化されているキュー・マネージャーをアクティブにします。
addMessageListener	オブジェクトを MQSeries Everyplace メッセージ・イベントの listener として登録します。
browseMessages	指定されたフィルターに一致する指定されたキューに存在するメッセージを含む MQeEnumeration を戻します。
browseMessagesAndLock	**browseMessages()** と同じ処理を行いますが、戻されたすべてのメッセージはキューにロックされたまま残ります。
checkActive	キュー・マネージャーがアクティブかどうかを示すブール値を戻します。
close	キュー・マネージャーをクローズします。
confirmGetMessage	以前の **getMessage()** 操作を確認します。
confirmPutMessage	以前の **putMessage()** 操作を確認します。
deleteMessage	メッセージを指定されたキューから除去します。
getMessage	メッセージを指定されたキューから戻します。
getName	固有のキュー・マネージャー名を戻します。
getReference	指定されたキュー・マネージャーへのオブジェクト参照を戻します。
putMessage	メッセージを指定されたキューに入れます。
removeMessageListener	MQSeries Everyplace メッセージ・イベントへのオブジェクト・サブスクリプションを取り消します。
triggerTransmission	保留メッセージの伝送開始をアプリケーションに許可します。

メソッド	目的
undo	このメソッドは、**putMessage()**、**getMessage()**、または **browseMessagesAndLock()** コマンドの実行時にエラーが発生した場合に使用するためのものです。
unlockMessage	事前に **browseMessagesAndLock()** 操作によってロックされているメッセージをアンロックします。
waitForMessage	**getMessage()** と同じ処理を行いますが、使用可能なメッセージが存在しない場合、キュー・マネージャーはメッセージが使用可能になるまで、指定の期間待機します。

MQeQueueManager activate

構文

1.
```
public void activate( MQeFields startupParameters ) throws Exception
```

2.
```
public void activate( String name ) throws Exception
```

記述

注: キュー・マネージャーを開始する前に、必要なすべての MQSeries Everyplace 別名を追加しておく必要があります。これを行う方法について、詳しくは、「*MQSeries Everyplace for Multiplatforms プログラミング・ガイド*」を参照してください。

このメソッドには次の 2 つのバージョンがあります。

1. 一方は、推奨されるバージョンです。これは、キュー・マネージャーの始動パラメーターを含む MQeFields オブジェクトを入力として取ります。次いで、キュー・マネージャーはすべてのサブコンポーネントを正しく初期化し、レジストリー内に保管されている情報を読み取ります。
2. このバージョンは、標準のキュー・マネージャー活動化プロシージャーを MQeQueueManager から拡張したクラスによってオーバーライドできるようにするためだけに用意されています。このメソッドは、キュー・マネージャー名を設定する以外の活動化プロシージャーを実行しません。

注: MQeQueueManager を拡張するクラスは、**activate()** メソッドを呼び出して、キュー・マネージャー名が正しく設定されるようにする必要があります。

パラメーター

startupParameters
キュー・マネージャーの始動パラメーターを含む MQeFields オブジェクト。

始動パラメーターには、**MQeQueueManager.QueueManager** (キュー・マネージャーをセットアップする) と **MQeQueueManager.Registry** (レジストリーをセットアップする) という 2 つのセクションが含まれていなければなりません。

MQeQueueManager.QueueManager

このセクションには、以下のものが含まれています。

MQeQueueManager.Name

キュー・マネージャーの固有名を含む ASCII ストリング。この名前は、以下の条件を満たしていなければなりません。

- 長さが 1 文字以上である
- ASCII 文字セット (つまり、20 より大きく 128 より小さい値の文字) に準拠している
- {}[]#():;,'"= という文字はいずれも使用できない
- キュー・マネージャー名の最初と最後の文字はピリオドにしない

ただし、MQSeries との互換性を保つため、キュー・マネージャー名は、最大長が 48 文字にすることをお勧めします。使用できる文字は以下のとおりです。

- 大文字 A 〜 Z
- 小文字 a 〜 z
- 数値 0 〜 9
- ピリオド (.)
- 下線 (_)
- パーセント記号 (%)

MQeQueueManager.Registry

このレジストリーは、存在して、以下のものが含まれていなければなりません。

MQeRegistry.LocalRegType

使用するレジストリーのタイプを含む ASCII ストリング。現在認識されているタイプは、**MQeRegistry.FileRegistry** と **MQeRegistry.PrivateRegistry** だけです。

注: いったんレジストリーが作成されたら、レジストリー・タイプを変更しないようお勧めします。レジストリー・タイプを変更すると、セキュア・キューが正しく機能しなくなる場合があります。

MQeRegistry.DirName

キュー・マネージャーのレジストリーへのパス名を含む ASCII ストリング。

専用レジストリーを使用するには、以下の値が必要です。

MQeRegistry.PIN

専用レジストリーの PIN を含む ASCII 値。

専用レジストリーを使用する場合で、それが 最小限の証明サーバー (「MQSeries Everyplace for Multiplatforms プログラミング・ガイド」の『Security』節を参照) に登録されている場合は、以下の値が必要です。

MQeRegistry.KeyRingPassword

レジストリーの秘密鍵を保護するために使用されるパスワードまたはパスフレーズを含む ASCII 値。

注: セキュリティーに関する理由により、*PIN* および *KeyRingPassword* は、キュー・マネージャーがアクティブになるとすぐに、始動パラメーターから削除されます。

他の 2 つのセクション、**MQeQueueManager.AppRunList** および **MQeQueueManager.CloseAppRunList** はオプションです。これらのセクションは、キュー・マネージャーがアクティブなときに close 要求を受け取った場合に呼び出される MQSeries Everyplace アプリケーションのリストを指定します。 (244ページの『MQeRunListInterface』を参照してください)

name キュー・マネージャーの名前を含む ASCII ストリング。

戻り値 なし

例外

MQeException	Except_QMgr_Activated
	Except_QMgr_AlreadyExists
	Except_QMgr_InvalidQMgrName
	Except_QMgr_NotConfigured
	Except_NotFound

例

```
class MyMQeApplication
{
  ...
  /* Create QueueManager startup parameters */
  MQeFields QMgrParams = new MQeFields();
  QMgrParams.PutAscii( MQeQueueManager.Name, "TestQMgr" );
  QMgrParams.PutAscii( MQeQueueManager.QueueStore, "MsgLog:c:¥¥TestQMgr" );

  /* Create Registry startup parameters */
  MQeFields RegParams = new MQeFields();
  RegParams.PutAscii( MQeQueueManager.RegType, MQeQueueManager.FileRegistry );
  RegParams.PutAscii( MQeQueueManager.Path, "MsgLog:c:¥¥TestQMgr¥¥Registry" );

  /* Combine the two sets of parameters into a single Fields object */
  MQeFields Params = new MQeFields();
  Params.PutFields( MQeQueueManager.QueueManager, QMgrParams );
  Params.PutFields( MQeQueueManager.Registry, RegParams );

  /* Instantiate 'null' Queue Manager */
  MQeQueueManager QMgr = new MQeQueueManager( );
  QMgr.Activate( Params ); /* Activate QMgr using parameters */
  ...
}
```

関連する関数

- **close()**

MQeQueueManager addMessageListener

構文

```
public void addMessageListener(  MQeMessageListenerInterface listener,
                                 String queueName,
                                 MQeFields filter ) throws Exception;
```

記述

このメソッドは、オブジェクトを、 *queueName* パラメーターで指定されたキューによって生成された、あらゆる MQeMessage イベントの listener として登録します。 listener は、ローカル・キューにのみ追加できます。

注: listen オブジェクトは MQeMessageListenerInterface を実装します。イベントは、このインターフェースで指定されたイベント・ハンドラー・メソッドによって処理されます。

メッセージ・フィールドから構成されるメッセージ・フィルター (たとえば、メッセージ ID または優先順位) を指定すれば、listen オブジェクトが、指定されたものと同じフィールドを含むメッセージに関するイベントだけを受信するようにできます。フィールドを指定しないと、イベントはキューのすべてのメッセージについて起動します。

パラメーター

listener サブスクリプションするオブジェクトの参照。

queueName listener が受信するイベントが入っているキューの名前を含む String。

params ヌル、またはメッセージ・フィールドを含む MQeFields オブジェクト。値をヌルに指定すると、 listener がキューにあるすべてのメッセージについてイベントを受け取ることを意味します。メッセージ・フィールドを含む MQeFields オブジェクトを指定する場合、リスナーは、フィルターに含まれるフィールドと一致するフィールドを持つメッセージに関係したイベントだけを受信します。

戻り値 なし

例外

MQeException Except_QMgr_NotActive

Except_QMgr_QdoesNotExist

例

```
class MyMQeApplication implements MQeMessageListenerInterface
{
  ...
  MQeFields filter = new MQeFields();   /* search parameters */
  filter.putByte( MQe.Msg_Priority,(byte)3);   /* only interested in  */
                                               /* msgs of priority 3 */
  ...
  /* add listener */
  MyQM.addMessageListener( this, "MyQueue", filter );
  ...
  /* Message arrived event handler */
  public void messageArrived( MQeMessageEvent msgEvent )
  {
    ...
    /* is it the Queue we are interested in?? */
    if ( msgEvent.getQueueName().equals("MyQueue") )
    {
      ...
    }
    ...
  }
}
```

関連する関数

- **removeMessageListener()**

MQeQueueManager browseMessages

構文

```
public MQeEnumeration browseMessages( String qmgrName,
                                      String queueName,
                                      MQeFields filter,
                                      MQeAttribute attribute,
                                      boolean justUID ) throws Exception;
```

記述

このメソッドは、指定されたキューで使用可能なメッセージの列挙型を戻します。メッセージはキューからは削除されません。キューは、ローカルまたはリモートのキュー・マネージャーに属することができます。

メッセージ・フィールド (たとえば、メッセージ ID や優先順位) で構成されるフィルターを指定することができます。このようにすると、一致するフィールドを持つメッセージだけが戻されます。

メッセージの列挙型を完全に戻すことは、システム・リソースの観点から見ると高価になります。それで、*justUID* パラメーターが `true` に設定されていれば、フィルターに一致するメッセージの固有の ID だけが戻されます。

列挙型で戻されたメッセージは、引き続き他の MQSeries Everyplace アプリケーションから見ることができます。したがって、列挙型に含まれるメッセージに対して以降の操作を実行する際、アプリケーションでは、列挙型が戻された後に他のアプリケーションがメッセージを処理した可能性があることを認識しておく必要があります。列挙型に含まれるメッセージをロックし、それによって他のアプリケーションがそのメッセージを処理することを防ぐには、**browseMessagesAndLock** メソッドを使用してください。

パラメーター

qmgrname
ブラウズするキューを保持するキュー・マネージャーの名前を含む String。値がヌルである場合、ローカルのキュー・マネージャーが使用されると想定されます。

queueName
ブラウズするキューの名前を含む String。

filter
ヌル、またはブラウズ実行時に使用するパラメーターを含む MQeFields オブジェクト。

attribute
メッセージ・レベルのセキュリティーを提供するために使用する MQeAttribute オブジェクト。

justUID
メッセージ内のすべてのフィールドを戻すか、*UID* 値だけを戻すかを示すブール値。

戻り値 ゼロ個以上の MQeMsgObject メッセージ・オブジェクトを含む MQeEnumeration。

例外

MQeException	Except_QMgr_NotActive
	Except_QMgr_InvalidQMgrName
	Except_QMgr_QDoesNotExist

他のさまざまな例外

例

```
class MyMQeApplication
{
  ...
  MQeEnumeration msgs  = null;
  byte[] msgId = MQe.asciiToByte(240999);
  byte[] correlId = MQe.asciiToByte("240999/2");

  try
  {
    /* setup parameters object for matching  */
    MQeFields filter = new MQeFields(); /* match against msgs */
    filter.putArrayOfByte( MQe.Msg_MsgID, msgId ); /* with this Msg Id */
    filter.putArrayOfByte( MQe.Msg_CorrelID,   /* & this Correl Id */
                           correlId );

    /* look at available messages  */
    msgs = qmgr.browseMessages( null, "MyQueue", filter, null, false );
    ...
    /* get this one and remove from queue  */
    MQeMsgObject msgObj = qmgr.getMessage( null, "MyQueue",
                                           (MQeFields)msgs.nextElement(),
                                           null, 0 );
  }
  catch ( MQeException e )
  {
    ...
  }
  ...
}
```

関連する関数

- **browseMessagesAndLock()**

MQeQueueManager browseMessagesAndLock

構文

```
public MQeEnumeration browseMessagesAndLock( String qmgrName,
                                             String queueName,
                                             MQeFields filter,
                                             MQeAttribute attribute,
                                             long confirmId,
                                             boolean justUID ) throws Exception;
```

記述 このメソッドは、指定されたキューで使用可能なメッセージの列挙型を戻しま

す。メッセージはキューからは削除されません。キューは、ローカルまたはリモートのキュー・マネージャーに属することができます。

メッセージ・フィールド (たとえば、メッセージ ID や優先順位) で構成されるフィルターを指定すれば、一致するフィールドを持つメッセージだけを戻すようにできます。

この操作によって戻されるメッセージは、キューでもロックされます。つまり、これらのメッセージは引き続きキューに存在していますが、アンロックされるまで、後続の操作で参照することはできません。

列挙型ブラウズの一部として、*lockID* が戻されます。 *lockID* により、ロックされたメッセージに対して操作を実行することができます。ただし、ロック ID は、その操作に渡されたメッセージ・フィルターの一部として指定されている必要があります。

各 *lockID* は固有なので、ブラウズやロック操作を行うたびに異なる ID が生成されます。 *lockID* は、ブラウズ操作によって戻されるすべてのメッセージに適用されます。

ロックされたメッセージに対して実行できる操作は以下のとおりです。

- **getMessage()**
- **deleteMessage()**
- **unlockMessage()**

注: waitForMessage は、ロックされたメッセージと一緒に使用しないでください。結果が予測できません。

メッセージの列挙型を完全に戻すと、システム・リソースの観点から見た場合に負荷が大きくなる可能性があります。そこで、*justUID* パラメーターが true に設定されていれば、フィルターに一致するメッセージの固有の ID だけが戻されます。

MQeAttribute オブジェクトを指定すると、メッセージのブラウズで、一致する属性によってメッセージ・レベルのセキュリティーを定義することができます。メッセージ・レベルのセキュリティーのレベルが異なるメッセージを含むキューをブラウズすると、未定義の結果になる可能性があります。

confirmID は、このコマンドの実行時にエラーが生じた場合に使用されます。エラーが生じうるのは、ロック ID がアプリケーションに戻されておらず、メッセージがまだターゲット・キューでロック状態になっている場合です。このメソッドで使用したのと同じ *confirmID* を **undo()** メソッドに渡せば、メッセージは以前の状態に復元されます。ブラウズおよびロック操作のたびに、固有の値を使用するようお勧めします。固有の値は、**MQe.uniqueValue()** メソッドを使用して生成できます。

パラメーター

qmgrName ブラウズするキューを保持するキュー・マネージャーの名前を含む String。値がヌルである場合、ローカルのキュー・マネージャーが使用されると想定されます。

queueName ブラウズするキューの名前を含む String。

filter ヌル、またはブラウズ実行時に使用するメッセージ・フィールドを含む MQeFields オブジェクト。

attribute メッセージ・レベルのセキュリティーを提供するために使用する MQeAttribute オブジェクト。

confirmId キュー・マネージャー障害の発生時に使用される long 値。アプリケーションでは、使用した値を保管しておき、障害発生時にメッセージをリセットするために使用する必要があります。

justUID メッセージ全体を戻すか、その *UID* だけを戻すかを示すブール値。

戻り値 ゼロ個以上の MQeMsgObject メッセージ・オブジェクトを含む MQeEnumeration。列挙型には、ロック ID も含まれます。これには、**getLockID()** メソッドを使用してアクセスできます。

例外

MQeException

Except_QMgr_NotActive

Except_QMgr_InvalidQMgrName

Except_QMgr_QDoesNotExist

他のさまざまな例外

例

```
class MyMQeApplication extends MQe
{
  ...
  MQeEnumeration msgs  = null;
  byte[] msgId = asciiToByte("240999");
  byte[] correlId = asciiToByte("240999/2");

  try
  {
    /* setup parameters object for matching  */
    MQeFields filter = new MQeFields(); /* match against msgs */
    filter.putArrayOfByte( MQe.Msg_MsgId, msgId ); /* with this Msg Id */
    filter.putArrayOfByte( MQe.Msg_CorrelId,   /* & this Correl Id */
                           correlId );
    /* look at available messages  */
    msgs = qmgr.browseMessagesAndLock( null, "MyQueue", filter, null, 0,
                                       false );
    long lockId = msgs.getLockId(); /* get Lock Id */

    filter.putLong( MQe.Msg_LockID, lockId ); /* Add lock Id */
```

```
    /* get the first locked message from queue  */
    MQeMsgObject msgObj = qmgr.getMessage( null, "MyQueue", filter, null, 0 );
  }
  catch ( MQeException e )
  {
    ...
  }
  ...
}
```

関連する関数

- **getMessage()**
- **waitForMessage()**
- **browseMessagesAndLock()**
- **unlockMessage()**
- **deleteMessage()**
- **undo()**

MQeQueueManager checkActive

構文

```
public boolean checkActive()
```

記述 このメソッドは、キュー・マネージャーがアクティブかどうかをアプリケーションが決定できるようにします。

パラメーター

なし

戻り値 キュー・マネージャーがアクティブかどうかを示すブール値。

例外 なし

例

```
class MyMQeApplication
{
  ...
  qmgr = new MQeQueueManager( startupParams );
  if ( qmgr.checkActive() ) /* verify that QMgr is active         */
  {
    ...
  }
  else
    throw new Exception( "Queue Manager not active" );
}
```

MQeQueueManager close

構文

```
public void close() throws MQeException
```

記述 このメソッドは、キュー・マネージャーをクローズします。これは、キュー・マネージャーの使用を完了したときに MQSeries Everyplace アプリケーションから呼び出す必要があります。

パラメーター

なし

戻り値 なし

例外

MQeException	Except_QMgr_NotActive

例

```
class MyMQeApplication
{
  ...
  try
  {
    qmgr.putMessage( null, "MyQueue", msgObj, null, 0 );
  }
  catch ( MQeException e )
  {
    ...
  }
  ...
  qmgr.close(); /* close QMgr */
}
```

関連する関数

- **activate()**

MQeQueueManager confirmGetMessage

構文

```
public void confirmGetMessage( String qmgrName,
                               String queueName,
                               MQeFields filter ) throws Exception
```

記述 このメソッドは、以前の **getMessage()** 操作でキューから取り出されたメッセージが正常に受け取られたかどうかを確認します。メッセージは、確認フローが受け取られるまでターゲット・キューでロックされたままです。

パラメーター

queueName	メッセージが保持されるキューの名前を含む String。
qmgrName	キューを保持するキュー・マネージャーの名前を含む String。値がヌルである場合、ローカルのキュー・マネージャーが使用されると想定されます。
filter	メッセージ・フィルターを含む MQeFields オブジェクト。操

作が成功するには、このフィルターに、メッセージの *UID* が含まれていなければなりません。

戻り値 なし

例外

MQeException Except_NotFound

注: この例外が発生するのは、すでに確認されたメッセージを確認しようとしたときです。アプリケーションがメッセージ取得の確認要求を再発行した場合、この例外は成功を示す戻りコードとして扱われます。

他のさまざまな例外

例

```
class MyMQeApplication
{
  ...
  /* generate a unique confirmId for this operation */
  long confirmId = MQe.uniqueValue();
  /* get next available msg - msg still locked on target queue */
  MQeMsgObject msg = qmgr.getMessage( "RemoteQMgr", "RemoteQueue", null,
   null, confirmId );
  /* confirm the successful Get */
  qmgr.confirmGetMessage( "RemoteQMgr", "RemoteQueue",
      msg.getMsgUIDFields() );
  ...
}
```

関連する関数

- **getMessage()**

MQeQueueManager confirmPutMessage

構文

```
public void confirmPutMessage( String qmgrName,
                               String queueName,
                               MQeFields filter ) throws Exception
```

記述 このメソッドは、以前に成功した **putMessage()** 操作の確認を行います。

パラメーター

queueName メッセージが保持されるキューの名前を含む String。

qmgrName キューを保持するキュー・マネージャーの名前を含む String。値がヌルである場合、ローカルのキュー・マネージャーが使用されると想定されます。

filter メッセージ・フィルターを含む MQeFields オブジェクト。操作が成功するには、フィルターにメッセージの固有の ID が含まれていなければなりません。

戻り値 なし

例外

MQeException Except_NotFound

注: この例外が発生するのは、すでに確認されたメッセージを確認しようとしたときです。アプリケーションが書き込みメッセージの確認要求を再発行した場合、この例外は成功を示す戻りコードとして扱われます。

他のさまざまな例外

例

```
class MyMQeApplication
{
  ...
  /* generate a unique confirmId for this operation */
  long confirmId = MQe.uniqueValue();
  qmgr.putMessage( "RemoteQMgr", "RemoteQueue", msg, null,
                   confirmId );
  /* confirm the put  */
    qmgr.confirmPutMessage( "RemoteQMgr", "RemoteQueue",
                            msg.getMsgUIDFields() );
  ...
}
```

関連する関数

- **putMessage()**

MQeQueueManager deleteMessage

構文

```
public void deleteMessage( String qmgrName,
                           String queueName,
                           MQeFields filter  ) throws MQeException
```

記述

このメソッドは、メッセージをキューから削除します。呼び出し元のアプリケーションにメッセージを戻すことはありません。

1 回の操作で削除できるメッセージは 1 つだけで、メッセージの *UID* (タイム・スタンプと起点キュー・マネージャー名) は必ず指定する必要があります。

キューは、ローカルまたはリモートの MQSeries Everyplace キュー・マネージャーに属することができます。

以前の操作 (たとえばブラウズ) によりロックされているメッセージは、メッセージ・フィルターに有効な *lockID* を組み込むことによって削除できます。

メッセージが使用可能でない場合、例外が戻されます。

パラメーター

queueName メッセージが保持されるキューの名前を含む String。

qmgrName キューを保持するキュー・マネージャーの名前を含む String。値がヌルである場合、ローカルのキュー・マネージャーが使用されると想定されます。

filter メッセージ・フィルターを含む MQeFields オブジェクト。操作が成功するには、フィルターにメッセージの *UID* が含まれていなければなりません。

戻り値 なし

例外

MQeException

- Except_QMgr_InvalidQName
- Except_QMgr_NotActive
- Except_QMgr_QDoesNotExist
- Except_QMgr_WrongType
- Except_NotFound
- Except_NotAllowed

他のさまざまな例外

例

```
class MyMQeApplication
{
  ...
  MQeEnumeration msgEnum;
  ...
  MQeFields filter = new MQeFields();
  filter.putArrayOfByte( MQe.Msg_MsgID, new byte[]{ 1,2,3,4 } );
  /* return all messages with a Message Id of 1234 */
  msgEnum = qmgr.browseMessages( null, "MyQueue", filter, null, false );
  /* delete all message with a Message Id of 1234 */
  while( msgEnum.hasMoreElements() )
    qmgr.deleteMessage( null, "MyQueue",
                        (MQeMsgObject)msgEnum.nextElement() );
  ...
}
```

関連する関数

- **waitForMessage()**
- **browseMessages()**
- **browseMessagesAndLock()**
- **putMessage()**
- **getMessage()**

MQeQueueManager getMessage

構文

```
public MQeMsgObject getMessage(  String qmgrName,
                                 String queueName,
                                 MQeFields filter,
                                 MQeAttribute attribute,
                                 long confirmId ) throws MQeException;
```

記述

このメソッドは、指定されたキューから利用可能なメッセージを戻します。そのメッセージはキューから除去されます。キューは、ローカルまたはリモートの MQSeries Everyplace キュー・マネージャーに属することができます。

メッセージ・フィルターが指定されない場合、キューで最初に使用可能なメッセージが戻されます。メッセージ・フィルターが指定された場合、フィルターに一致する最初に使用可能なメッセージが戻されます。

以前のブラウズ操作でロックされているメッセージは、メッセージのロックに使用された *lockID* をメッセージ・フィルターに含めることによって検索できます。

メッセージが使用できない場合、例外が戻されます。

保証されたメッセージ送達の使用は、*confirmID* パラメーターの値によって決まります。非ゼロ値を渡すと通常のメッセージが戻されますが、そのメッセージはロックされており、後続の確認が受け取られるまでターゲット・キューから除去されません。確認は、**confirmGetMessage()** メソッドによって発行できます。ゼロの値を渡すと、メッセージが戻され、そのメッセージはターゲット・キューから除去されますが、メッセージ送達は保証されません。

confirmID パラメーターは、このコマンドの実行時にエラーが生じた場合にも使用されます。障害が生じうるのは、ロック ID がアプリケーションに戻されておらず、メッセージがまだターゲット・キューでロック状態になっている場合です。取得操作で使用したのと同じ *confirmID* を undo メソッドに渡せば、メッセージは以前の状態に復元されます。取得操作のたびに、固有の値を使用するようお勧めします。固有の値は、**MQe.uniqueValue()** メソッドを使用して生成できます。

パラメーター

queueName 取得するメッセージが入っているキューの名前を含む String。

qmgrName キューを保持するキュー・マネージャーの名前を含む String。値がヌルである場合、ローカルのキュー・マネージャーが使用されると想定されます。

filter ヌル、またはメッセージ・フィルターを含む MQeFields オブジェクト。

attribute メッセージ・レベルのセキュリティーを提供するために使用する MQeAttribute オブジェクト。提供された属性は、このメソッドで戻されるメッセージに付加された属性と一致しなければなりません。このようになっていないと、メッセージが消失する可能性があります。

confirmId 保証されたメッセージ送達を使用するかどうかを示す long 値。非ゼロ値の場合、メッセージはターゲット・キューから除去されません。これが行われるのは以降の確認フローのときです。ゼロの値の場合、メッセージはターゲット・キューから除去されます。

戻り値 指定されたキューから取得したメッセージを含む MQeMsgObject。

例外

MQeException

`Except_QMgr_NotActive`

`Except_QMgr_InvalidQName`

`Except_QMgr_QDoesNotExist`

`Except_QMgr_WrongQType`

`Except_Q_NoMatchingMsg`

`Except_NotFound`

他のさまざまな例外

例 例 1- 単純な取得操作。メッセージ・フィルターなし

```
class MyMQeApplication
{
  ...
  try
  {
    /* get 1st available message on the queue */
    MQeMsgObject myMsgObject = qmgr.getMessage( null, "MyQueue", null, null,
                                                0 );
  }
  catch ( MQeException e )
  {
    ...
  }
  ...
}
```

例 2- ブラウズおよび取得操作

```
class MyMQeApplication
{
  ...
  /* Lock all msgs on this queue /*
  MQeEnumeration msgEnum = qmgr.browseMessagesAndLock( null, "MyQueue",
                                                       null, null, 0, false );
  long lockId = msgEnum.getLockId(); /* get the Lock Id */
  MQeFields filter = new MQeFields(); /* create a msg filter */
  filter.putLong( MQe.Msg_LockID, lockId ); /* add lock Id */
  /* get the 1st locked message on the queue */
  MQeMsgObject msgObj = qmgr.getMessage( null, "MyQueue", filter, null, 0 );
  ...
}
```

例 3- 保証されたメッセージ送達を使用する取得操作

```
class MyMQeApplication
{
  ...
  /* generate a unique confirmId for this operation */
  long confirmId = MQe.uniqueValue();
  /* get next available msg - msg remains locked on the target queue */
  MQeMsgObject msg = qmgr.getMessage( "RemoteQMgr", "RemoteQueue", null,
                                      null, confirmId );
  /* confirm the successful Get */
  qmgr.confirmGetMessage( "RemoteQMgr", "RemoteQueue",
                          msg.getMsgUIDFields() );
  ...
}
```

関連する関数

- **waitForMessage()**
- **browseMessages()**
- **browseMessagesAndLock()**
- **putMessage()**
- **deleteMessage()**
- **confirmGetMessage()**
- **undo()**

MQeQueueManager getName

構文

```
public String getName();
```

記述 このメソッドは、このキュー・マネージャーの名前を戻します。

注: すべてのキュー・マネージャー名を MQSeries Everyplace ネットワーク内で固有のものにするよう強くお勧めします。

パラメーター

なし

戻り値 キュー・マネージャーの名前を含む String。

例外 なし

例

```
class MyMQeApplication
{
  ...
  String qmgrName = qmgr.getName();
  ...
}
```

MQeQueueManager getReference

構文

```
public static MQeQueueManager getReference( String qmgrName) throws MQeException
```

記述 このメソッドは、インスタンス化されたキュー・マネージャーへのオブジェクト参照を取得するために使用します。

パラメーター

qmgrName キュー・マネージャーの名前を含む String。

戻り値 MQeQueueManager オブジェクト。

例外

MQeException Except_QMgr_InvalidQMgrName

例

```
class MyMQeApplication
{
  ...
  MQeQueueManager qmgr = null;
  ...
  /* Obtain a reference to "MyQMgr" Queue Manager /*
  qmgr = MQeQueueManager.getReference( "MyQMgr" );
  /* Put a message  /*
  qmgr.putMessage( null, "DestQ", Msg, null, 0 );
  ...
}
```

MQeQueueManager putMessage

構文

```
public void putMessage( String qmgrName,
                        String queueName,
                        MQeMsgObject msg,
                        MQeAttribute attribute,
                        long confirmId ) throws Exception;
```

記述 このメソッドは、指定されたメッセージを指定されたキューに入れます。このキューは、ローカルまたはリモートのキュー・マネージャーに属することができます。

リモート・キューへの書き込みは、リモート・キューがローカルのキュー・マネージャーで定義されている仕方に応じて、即座に、または少し後の時刻に行われます。

リモート・キューが同期として定義されている場合、ネットワークでのメッセージ伝送は即座に行われます。

リモート・キューが非同期として定義されている場合、メッセージは、ローカルのキュー・マネージャー内に格納されます。キュー・マネージャーのルールによって保留メッセージを伝送する時機であると判断されるか、または **triggerTransmission()** メソッドによってキュー・マネージャーが起動されるまで、メッセージは、ローカルのキー・マネージャー内に残されます。

ローカルのキュー・マネージャーは、リモート・キューの定義を保持していない場合に、キューに同期的にアクセスしようとします。

保証されたメッセージ送達は、*confirmID* パラメーターの値によって決まります。非ゼロ値を渡すと通常のメッセージが伝送されますが、そのメッセージは、後続の確認が受け取られるまでターゲット・キューにロックされています。ゼロの値を渡すと、以降の確認も必要なくメッセージは伝送されますが、メッセージの伝送は保証されません。

confirmID は、このコマンドの実行時にエラーが生じた場合にも使用されます。書き込み操作で使用したのと同じ confirmID を undo メソッドに渡せば、未確認のメッセージはターゲット・キューから除去されます。書き込み操作のたびに、固有の値を使用するようお勧めします。固有の値は、**MQe.uniqueValue()** メソッドを使用して生成できます。

メッセージは、メッセージ・レベルのセキュリティーを使用して保護できます (MQSeries Everyplace のセキュリティーについては、「*MQSeries Everyplace for Multiplatforms プログラミング・ガイド*」を参照)。セキュリティーは、MQeAdminQueueAdminMsg オブジェクト、またはこの下位オブジェクトの 1 つを指定することによって定義されます。この属性は、メッセージ書き込み要求の前にメッセージに付加することができます。または、属性パラメーターを使用して、メッセージ・レベルのセキュリティーを使用することを指定することができます。

属性パラメーターがヌルでない場合、この値は、メッセージ書き込み要求の前にメッセージに付加された属性をオーバーライドします。属性パラメーターがヌルの場合、メッセージの送信に対する影響はありません。

パラメーター

queueName メッセージが入れられるキューの名前を含む String。

qmgrName 指定されたキューが属するリモート・キュー・マネージャーの名前を含む String。値がヌルである場合、ローカルのキュー・マネージャーが使用されると想定されます。

msg メッセージを含む MQeMsgObject。

attribute MQeAdminQueueAdminMsg オブジェクトまたは下位オブジェクト、もしくはヌル。

ヌルの場合、このパラメーターはメッセージの送信に影響しません。

ここで指定された属性は、以前にメッセージに関連付けられた任意の属性をオーバーライドし、さらに、以前の **MQeFields.setAttribute()** または **MQeMsgObject.setAttribute()** メソッドの呼び出しを使用したメッセージ内の MQeFields データをオーバーライドします。

MQeMAttribute または MQeMTrustAttribute を渡せば、メッセージ・レベルのセキュリティー操作を実行できます。

confirmId 保証されたメッセージ送達を使用するかどうかを示す long 値。非ゼロ値の場合、メッセージはターゲット・キューでロックされ、以降の確認フローまで見えなくなります。ゼロの値の場合、以降の確認も必要なくメッセージは伝送されます。

戻り値 なし。

例外

MQeException

- `Except_QMgr_InvalidQName`
- `Except_QMgr_NotActive`
- `Except_QMgr_QDoesNotExist`
- `Except_Duplicate`

他のさまざまな例外

例 例 1- 単純な書き込み操作

```
class MyMQeApplication
{
  ...
  try
  {
    qmgr.putMessage( null, "MyQueue", msgObj, /* simple put */
                     null, 0 );
  }
  catch ( MQeException e )
  {
```

```
    ...
  }
  ...
}
```

例 2- 保証されたメッセージ送達を使用する書き込み操作

```
class MyMQeApplication
{
  ...
  /* generate a unique confirmId for this operation */
  long confirmId = MQe.uniqueValue();
    qmgr.putMessage( "RemoteQMgr", "RemoteQueue", msg, null, confirmId );
  /* confirm the put  */
    qmgr.confirmPutMessage( "RemoteQMgr", "RemoteQueue",
                            msg.getMsgUIDFields() );
  ...
}
```

関連する関数

- **getMessage()**
- **waitForMessage()**
- **confirmPutMessage()**
- **undo()**

MQeQueueManager removeMessageListener

構文

```
public void removeMessageListener( MQeMessageListenerInterface listener,
                                   String queueName,
                                   MQeFields filter  ) throws MQeException
```

記述 このメソッドは、*queueName* で指定されたキューで生成された MQSeries Everyplace メッセージ・イベントへのオブジェクトのサブスクリプションを除去します。 listener は、ローカル・キューにのみ存在できます。

注: listen オブジェクトは MQeMessageListenerInterface を実装します。

オプションのメッセージ・フィルターが設定されていると、オブジェクトのサブスクリプションは、フィルターで指定されたものと同じフィールドを含むメッセージが関係するイベントについてのみ除去されます。フィルターがヌルであれば、すべてのメッセージに関係するイベントについて、オブジェクトのサブスクリプションは除去されます。

パラメーター

listener サブスクリプションするオブジェクトの参照。

queueName リスナーが受信するイベントが入っているキューの名前を含む String。

filter ヌル、またはメッセージ・フィルターを含む MQeFields オブジェクト。

戻り値 なし

例外

MQeException	Except_QMgr_NotActive
	Except_QMgr_InvalidQName
	Except_QMgr_QDoesNotExist

例

```
class MyMQeApplication implements MQeMessageListenerInterface
{
  ...
  /* remove the 'all messages' listener for this queue */
  qmgr.removeMessageListener( this, "MY.QUEUE", null );
  ...
}
```

関連する関数

addMessageListener

MQeQueueManager triggerTransmission

構文

```
public void triggerTransmission() throws Exception
```

記述 このメソッドは、保留メッセージの伝送を試みます。

保留メッセージとは、リモート・キュー・マネージャーへの伝送を待機しているメッセージのことです。通常、保留メッセージの伝送は、キュー・マネージャーのルールによって処理されますが、このメソッドを使用すれば、保留メッセージの伝送をアプリケーションの都合がよい時刻に行うことができます。

さらに、このメソッドは定義されているすべてのホーム・サーバー・キューを起動します。これらのキューは、ホーム・サーバーからメッセージを収集しようとします。

このメソッドは、**MQeQueueManagerRule.triggerTransmission()** ルールの操作をオーバーライドしますが、 **MQeQueueManagerRule.transmi()** ルールを呼び出します。

パラメーター

なし

戻り値 なし

例外

MQeException	Except_BadRequest

```
                                    Except_QMgr_NotActive

                                    Except_QMgr_QDoesNotExist
```

他のさまざまな例外

例

```
class MyMQeApplication
{
  ...
  try
  {
    if ( timeToTransmit() ) /* application decides it's time to  */
      qmgr.triggerTransmission(); /* transmit */
  }
  catch ( MQeException e )
  {
    if ( e.Code() != Except_QMgr_Busy )
      throw e;
  }
  ...
}
```

MQeQueueManager undo

構文

```
public void undo( String qmgrName,
                  String queueName,
                  long confirmId ) throws Exception
```

記述 このメソッドは、**put()**、**get()**、または **browseAndLock()** コマンドの実行時にエラーが発生した場合に使用するためのものです。エラーにより、メッセージが未確認またはロック状態でターゲット・キューに残ることがあります。このメソッドは、メッセージを失敗した操作の前の状態 (ロックまたはアンロック) にリセットします。未確認の書き込み操作の場合は、メッセージを削除します。

メッセージをリセットするには、失敗した操作で使用された *confirmID* を指定する必要があります。 *confirmID* は、各メッセージごとに固有にすることをお勧めします。固有の値は、**MQe.uniqueValue()** メソッドを使用して生成できます。

パラメーター

qmgrName キューを保持するキュー・マネージャーの名前を含む String。値がヌルである場合、ローカルのキュー・マネージャーが使用されるキュー・マネージャーとして想定されます。

queueName ロックされたメッセージを保持するキューの名前を含む String。

confirmId 失敗した操作で使用された *confirmID* と同じ long 値。

戻り値 なし。

例外

MQeException	Except_QMgr_NotActive
	Except_QMgr_InvalidQName
	Except_QMgr_QDoesNotExist
	Except_Q_NoMatchingMsg
	Except_NotAllowed

他のさまざまな例外

例

```
class MyMQeApplication
{
  ...
  /* generate a unique confirmId for this operation */
  long confirmId = MQe.uniqueValue();
  try
  {
    qmgr.putMessage( "RemoteQMgr", "RemoteQueue", msg, null, confirmId );
    qmgr.confirmPutMessage( "RemoteQMgr", "RemoteQueue",
                            msg.getMsgUIDFields() );
  }
  catch ( Exception e )
  {
    /* Give the remote Queue Manager time to recover from error */
    Thread.sleep( 30000 );
    /* Remote Queue Manager failure - undo the put message */
    qmgr.undo("RemoteQMgr", "RemoteQueue", confirmId );
  }
  ...
}
```

関連する関数

- **browseMessagesAndLock()**
- **getMessage()**
- **putMessage()**

MQeQueueManager unlockMessage

構文

```
public void unlockMessage( String qmgrName,
                           String queueName,
                           MQeFields filter ) throws Exception
```

記述 このメソッドは、事前にロックされたメッセージをアンロックします。メッセージは、再びすべてのアプリケーションから見えるようになります。一度にアンロックできるメッセージは 1 つだけで、メッセージの *UID* (タイム・スタンプと起点キュー・マネージャー名) と *lockID* の両方を指定する必要があります。

キューは、ローカルまたはリモートのキュー・マネージャーに属することができます。

メッセージが使用可能でない場合、例外が戻されます。

このメソッドは通常、 **browseMessagesAndLock()** メソッドと組み合わせて使用します。

パラメーター

qmgrName キューを保持するキュー・マネージャーの名前を含む String。値がヌルである場合、ローカルのキュー・マネージャーが使用されると想定されます。

queueName ロックされたメッセージを保持するキューの名前を含む String。

filter メッセージ・フィルターを含む MQeFields オブジェクト。操作が成功するには、これにメッセージの固有の ID と操作のロック ID が含まれていなければなりません。

戻り値 なし

例外

MQeException

- Except_QMgr_NotActive
- Except_QMgr_InvalidQName
- Except_QMgr_QDoesNotExist
- Except_Q_NoMatchingMsg
- Except_NotAllowed

他のさまざまな例外

例

```
class MyMQeApplication
{
  ...
  MQeEnumeration msgEnum;
  ...
  /* lock all msgs on queue */
  msgEnum = qmgr.browseMessagesAndLock( null, "MyQueue", null, null, 0,
                                        false );
  long lockID = msgEnum.getLockId(); /* get lockID */
  while( msgEnum.hasMoreElements() )
  {
    MQeFields msgFields = (MQeFields)msgEnum.nextElement();
    String msgID = byteToAscii( msgFields.getArrayOfByte( MQe.Msg_MsgID ) );
    /* Unlock all messages with an ID of 1234 */
    if ( msgID.equals("1234") )
    {
      msgFields.putLong( MQe.Msg_LockID, lockID );
      qmgr.unlockMessage( null, "MyQueue", msgFields );
```

```
        }
      }
      ...
    }
```

関連する関数

- **browseMessageAndLock**

MQeQueueManager waitForMessage

構文

```
public MQeMsgObject waitForMessage( String qmgrName,
                                    String queueName,
                                    MQeFields filter,
                                    MQeAttribute attribute,
                                    long confirmId,
                                    int milliseconds ) throws MQeException;
```

記述 このメソッドは、 **getMessage** と同じ処理を行います。ただし、使用可能なメッセージが存在しない場合、このメソッドは *milliseconds* で指定された期間が過ぎるまで待機します。この期間が過ぎてもメッセージが使用できない場合、例外が戻されます。

パラメーター

qmgrName キューを保持するキュー・マネージャーの名前を含む String。値がヌルである場合、ローカルのキュー・マネージャーが使用されると想定されます。

queueName メッセージの取得先となる MQSeries Everyplace キューの名前を含むストリング。

filter ヌル、またはメッセージ・フィルターを含む MQeFields オブジェクト。

attribute メッセージ・レベルのセキュリティーを提供するために使用する MQeAttribute オブジェクト。

confirmId 保証されたメッセージ送達を使用するかどうかを示す long 値。非ゼロ値の場合、メッセージはターゲット・キューから除去されません。これが行われるのは以降の確認フローのときです。ゼロの値の場合、メッセージはターゲット・キューから除去されます。

milliseconds メッセージが使用可能になるのを待機する期間 (ミリ秒単位)。

戻り値 指定されたキューから取得されたメッセージを含む MQeMsgObject。

例外

MQeException `Except_QMgr_NotActive`

`Except_QMgr_InvalidQName`

Except_QMgr_QDoesNotExist

Except_Q_NoMatchingMsg

Except_Q_NoMsgAvailable

他のさまざまな例外

例

```
class MyMQeApplication extends MQe
{
  ...
  String MsgId = "260399";
  String CorrelId = "260399/2";
  ...
  /* set up a parameters object to match with */
  /* only interested in msgs*/
  MQeFields filter = new MQeFields();
  /* with this message Id*/
  filter.putArrayOfByte( MQe.Msg_MsgID, asciiToByte( MsgId ) );
  /* & this correlation Id */
  filter.putArrayOfByte ( MQe.Msg_CorrelID, asciiToByte( CorrelId ) );
  ...
  /* wait 10 seconds for a msg to arrive */
  MQeMsgObject msgObj = qmgr.waitForMessage( null, "MyQueue", filter,
          null, 0, 10000 );
  ...
}
```

関連する関数

- **getMessage()**

MQeQueueManagerConfigure

このクラスは、キュー・マネージャーを構成するときに使います。このクラスを使い、キュー・マネージャーとそのデフォルト・キューを作成したり削除します。

パッケージ **com.ibm.mqe**

コンストラクター

MQeQueueManagerConfigure

構文

```
1. public MQeQueueManagerConfigure( )
2.
   public MQeQueueManagerConfigure( MQeFields startupParameters ) throws Exception
3. public MQeQueueManagerConfigure( MQeFields startupParameters,
                                    String qStore ) throws Exception
```

説明 コンストラクターは、キュー・マネージャー構成オブジェクトをインスタンス化します。コンストラクターには、以下の 3 つの形式があります。

1. この形式は、動的ロード用に設計されています。動的ロードの後に **activate()** を呼び出す必要があります。
2. この形式は、キュー・マネージャーを削除するときにだけ使えます。
3. この形式は、キュー・マネージャーを作成または削除するときに使えます。

パラメーター

startupParameters
キュー・マネージャーの初期設定パラメーターを含む MQeFields オブジェクト。これらについては、MQeQueueManager startupParameters で説明します。

qStore
MQSeries Everyplace アダプターを示すストリング。これは、キュー・ストアにアクセスするために使用されるもので、この後にはコロン文字が続き、さらに、標準デフォルト・キューが格納されている場所が続きます。 これは、キュー・マネージャーを作成する予定の場合に指定する必要があります。キュー・マネージャーを削除する予定の場合には、このパラメーターはヌルにできます。

戻り値 なし

例外 例外 - キュー・マネージャー構成オブジェクトの開始に問題があれば示されます。

例

```
MQeQueueManagerConfigure qmConfig1;
qmConfig1 = new MQeQueueManagerConfigure();
```

```
try
{
 MQeQueueManagerConfigure qmConfig2;
 MQeFields parms = new MQeFields();
 // initialize the parameters
  ...
 qmConfig2 = new MQeQueueManagerConfigure( parms );
}
catch (Exception e)
{ ... }
try
{
 MQeQueueManagerConfigure qmConfig3;
 MQeFields parms = new MQeFields();
 // initialize the parameters
 ...
 qmConfig3 = new MQeQueueManagerConfigure( parms,"MsgLog:" + "qmName" + File.separator +
                                           "Queues" + File.separator );
}
catch (Exception e)
{ ... }
```

メソッド

メソッド	目的
activate	構成オブジェクトをアクティブにします。
close	構成オブジェクトをクローズします。
defineDefaultAdminQueue	レジストリーに標準デフォルトの*管理キュー* を定義します。
defineDefaultAdminReplyQueue	レジストリーに標準デフォルトの*管理応答キュー* を定義します。
defineDefaultDeadLetterQueue	レジストリーに標準デフォルトの*送達不能キュー* を定義します。
defineDefaultSystemQueue	レジストリーに標準デフォルトのローカル・キューを定義します。
defineQueueManager	レジストリーに標準のキュー・マネージャーを定義します。
deleteAdminQueueDefinition	レジストリーから*管理キュー* を削除します。
deleteAdminReplyQueueDefinition	レジストリーから*管理応答キュー* を削除します。
deleteDeadLetterQueueDefinition	レジストリーから*送達不能キュー* を削除します。
deleteQueueManagerDefinition	レジストリーからキュー・マネージャーを削除します。
deleteStandardQMDefinitions	レジストリーからキュー・マネージャーと標準のキューを削除します。
deleteSystemQueueDefinition	レジストリーから標準デフォルトのローカル・キューを削除します。
queueManagerExists	レジストリーにキュー・マネージャーがあるかどうかを調べます。
setChannelTimeout	キュー・マネージャーのチャネル・タイムアウト値を設定します。
setChnlAttributeRuleName	キュー・マネージャーのチャネル属性ルールの名前を設定します。

メソッド	目的
setDescription	キュー・マネージャーの説明を設定します。

MQeQueueManagerConfigure activate

構文

```
public void activate( MQeFields startupParameters, String qStore ) throws Exception
```

説明 このメソッドは、キュー・マネージャーを構成する準備ができたオブジェクトを初期設定します。

パラメーター

startupParameters
キュー・マネージャーの初期設定パラメーターを含む MQeFields オブジェクト。これらについては、MQeQueueManager startupParameters で説明します。

qStore
MQSeries Everyplace アダプターを示すストリング。これは、キュー・ストアにアクセスするために使用されるもので、この後にはコロン文字が続き、さらに、標準デフォルト・キューが格納されている場所が続きます。これは、キュー・マネージャーを作成する予定の場合に指定する必要があります。キュー・マネージャーを削除する予定の場合には、このパラメーターは、ヌルにできます。

戻り値 なし

例外 例外 - オブジェクトの開始に問題があれば示されます。

例

```
try
{
 MQeQueueManagerConfigure qmConfig;
 MQeFields parms = new MQeFields();
 // initialize the parameters
 ...
 qmConfig = new MQeQueueManagerConfigure( );
 qmConfig.activate( parms, "MsgLog:" + "qmName" + File.separator +
                    "Queues" + File.separator );
}
catch (Exception e)
{ ... }
```

MQeQueueManagerConfigure close

構文

```
public void close( )
```

説明 このメソッドは、構成オブジェクトをクローズします。オブジェクトのクロー

ズ後に使おうとすると、例外が生じます。構成オブジェクトは、キュー・マネージャーそのものをアクティブにする前に、クローズしておく必要があります。

パラメーター

なし

戻り値 なし

例外 なし

例

```
try
{
 MQeQueueManagerConfigure qmConfig;
 MQeFields parms = new MQeFields();
 // initialize the parameters
 ...
 qmConfig = new MQeQueueManagerConfigure( parms, "qmName" + File.separator +
                                                 "Queues" + File.separator );
 ...
 qmConfig.close();
}
catch (Exception e)
{ ... }
```

MQeQueueManagerConfigure defineDefaultAdminQueue

構文

```
public void void defineDefaultAdminQueue(String desc) throws Exception
```

説明 このメソッドは、キュー・マネージャーのレジストリーに標準の管理キュー を定義します。 この説明のパラメーターはオプションです。 キューそのものは、実行中のキュー・マネージャーから初めてアクセスされるときに作成されます。キューがすでに存在していると、例外が生じます。

パラメーター

desc 管理キューの説明を含む、オプションのストリング。

戻り値 なし

例外

MQeException MQeQueueManagerConfigure オブジェクトがアクティブにされていない場合、またはキュー・マネージャーのレジストリーにすでにキューがある場合に出されます。

例外 他のエラーの場合に示されます。

例

```
try
{
 MQeQueueManagerConfigure qmConfig;
```

```
 MQeFields parms = new MQeFields();
 // initialize the parameters
 ...
 qmConfig = new MQeQueueManagerConfigure( parms, "qmName" + File.separator +
                                                 "Queues" + File.separator );
 qmConfig.defineDefaultAdminQueue();
 ...
 qmConfig.close();
}
catch (Exception e)
{ ... }
```

MQeQueueManagerConfigure defineDefaultAdminReplyQueue

構文

```
public void defineDefaultAdminReplyQueue (String desc ) throws Exception
```

説明 このメソッドは、キュー・マネージャーのレジストリーに標準の*管理応答キュー* を定義します。 この説明のパラメーターはオプションです。 キューそのものは、実行中のキュー・マネージャーから初めてアクセスされるときに作成されます。キューがすでに存在していると、例外が生じます。

パラメーター

desc 管理応答キューの説明を含む、オプションのストリング。

戻り値 なし

例外

MQeException MQeQueueManagerConfigure オブジェクトがアクティブにされていない場合、またはキュー・マネージャーのレジストリーにすでにキューがある場合に出されます。

例外 他のエラーの場合に示されます。

例

```
try
{
 MQeQueueManagerConfigure qmConfig;
 MQeFields parms = new MQeFields();
 // initialize the parameters
 ...
 qmConfig = new MQeQueueManagerConfigure( parms, "qmName" + File.separator +
                                                 "Queues" + File.separator );
 qmConfig.defineDefaultAdminReplyQueue();
 ...
 qmConfig.close();
}
catch (Exception e)
{ ... }
```

MQeQueueManagerConfigure defineDefaultDeadLetterQueue

構文

```
public void defineDefaultDeadLetterQueue (String desc ) throws Exception
```

説明 このメソッドは、キュー・マネージャーのレジストリーに標準の送達不能キュー を定義します。 この説明のパラメーターはオプションです。 キューそのものは、実行中のキュー・マネージャーから初めてアクセスされるときに作成されます。キューがすでに存在していると、例外が生じます。

パラメーター

desc 送達不能キューの説明を含む、オプションのストリング。

戻り値 なし

例外

MQeException MQeQueueManagerConfigure オブジェクトがアクティブにされていない場合、またはキュー・マネージャーのレジストリーにすでにキューがある場合に示されます。

例外 他のエラーの場合に示されます。

例

```
try
{
 MQeQueueManagerConfigure qmConfig;
 MQeFields parms = new MQeFields();
 // initialize the parameters
 ...
 qmConfig = new MQeQueueManagerConfigure( parms, "qmName" + File.separator +
                                                 "Queues" + File.separator );
 qmConfig.defineDefaultDeadLetterQueue();
 qmConfig.close();
}
catch (Exception e)
{ ... }
```

MQeQueueManagerConfigure defineDefaultSystemQueue

構文

```
public void defineDefaultSystemQueue(String desc ) throws Exception
```

説明 このメソッドは、キュー・マネージャーのレジストリーに、SYSTEM.DEFAULT.LOCAL.QUEUE という標準のローカル・キューを定義します。この説明のパラメーターはオプションです。キューそのものは、実行中のキュー・マネージャーから初めてアクセスされるときに作成されます。キューがすでに存在していると、例外が生じます。

パラメーター

desc デフォルトのシステム・キューの説明を含む、オプションのストリング。

戻り値 なし

例外

MQeException MQeQueueManagerConfigure オブジェクトがアクティブにされていない場合、またはキュー・マネージャーのレジストリーにすでにキューがある場合に示されます。

例外 他のエラーの場合に示されます。

例

```
try
{
 MQeQueueManagerConfigure qmConfig;
 MQeFields parms = new MQeFields();
 // initialize the parameters
 ...
 qmConfig = new MQeQueueManagerConfigure( parms, "qmName" + File.separator +
                                                 "Queues" + File.separator );
 qmConfig.defineDefaultSystemQueue();
 ...
 qmConfig.close();
}
catch (Exception e)
{ ... }
```

MQeQueueManagerConfigure defineQueueManager

構文

```
public void defineQueueManager( ) throws Exception
```

説明 このメソッドは、レジストリーにキュー・マネージャーの定義を作成します。これは、キュー・マネージャーそのものをアクティブにする前に行う必要があります。キュー・マネージャーの定義がすでに存在していると、例外が生じます。

パラメーター
なし

戻り値 なし

例外

MQeException MQeQueueManagerConfigure オブジェクトがアクティブにされていない場合、またはレジストリーにすでにキュー・マネージャーの定義がある場合に出されます。

例外 他のエラーの場合に示されます。

例

```
try
{
 MQeQueueManagerConfigure qmConfig;
 MQeFields parms = new MQeFields();
 // initialize the parameters
 ...
 qmConfig = new MQeQueueManagerConfigure( parms, "qmName" + File.separator +
```

```
                                                "Queues" + File.separator );
 qmConfig.setDescription( "queue manager for " + qmName );
 qmConfig.defineQueueManager();
 qmConfig.close();
}
catch (Exception e)
{ ... }
```

MQeQueueManagerConfigure deleteAdminQueueDefinition

構文

```
public void deleteAdminQueueDefinition( ) throws Exception
```

説明 このメソッドは、キュー・マネージャーのレジストリーから標準の管理キューの定義を削除します。定義が存在していなければ、エラーは発生しません。キューそのものが削除されることはありません。

キューがレジストリーに定義されていなければ、キューにアクセスできません。定義は、**defineDefaultAdminQueue()** を使って再作成することができます。

パラメーター

なし

戻り値 なし

例外

MQeException	MQeQueueManagerConfigure オブジェクトがアクティブにされていない場合、またはレジストリー項目の削除でエラーが生じる場合に出されます。

例

```
try
{
 MQeQueueManagerConfigure qmConfig;
 MQeFields parms = new MQeFields();
 // initialize the parameters
 ...
 qmConfig = new MQeQueueManagerConfigure( parms, null );
 qmConfig.deleteAdminQueueDefinition();
 ...
 qmConfig.close();
}
catch (Exception e)
{ ... }
```

MQeQueueManagerConfigure deleteAdminReplyQueueDefinition

構文

```
public void deleteAdminReplyQueueDefinition ( ) throws Exception
```

説明 このメソッドは、キュー・マネージャーのレジストリーから標準の管理応答キュー の定義を削除します。定義が存在していなければ、エラーは発生しません。キューそのものが削除されることはありません。

キューがレジストリーに定義されていなければ、キューにアクセスできません。定義は、**defineDefaultAdminReplyQueue()** を使って再作成することができます。

パラメーター

なし

戻り値 なし

例外

MQeException	**MQeQueueManagerConfigure** オブジェクトがアクティブにされていない場合、またはレジストリー項目の削除でエラーが生じる場合に示されます。

例

```
try
{
 MQeQueueManagerConfigure qmConfig;
 MQeFields parms = new MQeFields();
 // initialize the parameters
 ...
 qmConfig = new MQeQueueManagerConfigure( parms, null );
 qmConfig.deleteAdminReplyQueueDefinition();
 ...
 qmConfig.close();
}
catch (Exception e)
{ ... }
```

MQeQueueManagerConfigure deleteDeadLetterQueueDefinition

構文

```
public void deleteDeadLetterQueueDefinition ( ) throws Exception
```

説明 これは、キュー・マネージャーのレジストリーから標準の送達不能キュー の定義を削除します。定義が存在していなければ、エラーは発生しません。キューそのものが削除されることはありません。

キューがレジストリーに定義されていなければ、キューにアクセスできません。定義は、**defineDefaultDeadLetterQueue()** を使って再作成することができます。

パラメーター

なし

戻り値 なし

例外

MQeException MQeQueueManagerConfigure オブジェクトがアクティブにされていない場合、またはレジストリー項目の削除でエラーが生じる場合に示されます。

例

```
try
{
 MQeQueueManagerConfigure qmConfig;
 MQeFields parms = new MQeFields();
 // initialize the parameters
 ...
 qmConfig = new MQeQueueManagerConfigure( parms, null );
 qmConfig.deleteDeadLetterQueueDefinition();
 ...
 qmConfig.close();
}
catch (Exception e)
{ ... }
```

MQeQueueManagerConfigure deleteQueueManagerDefinition

構文

```
public void deleteQueueManagerDefinition ( ) throws Exception
```

説明 このメソッドは、レジストリーからキュー・マネージャーの定義を削除します。定義が存在していなければ、エラーは発生しません。

キューがレジストリーに定義されていなければ、キューにアクセスできません。定義は、**defineQueueManager()** を使って再作成することができます。

パラメーター
なし

戻り値 なし

例外

MQeException MQeQueueManagerConfigure オブジェクトがアクティブにされていない場合、またはレジストリー項目の削除でエラーが生じる場合に出されます。

例

```
try
{
 MQeQueueManagerConfigure qmConfig;
 MQeFields parms = new MQeFields();
 // initialize the parameters
 ...
```

```
 qmConfig = new MQeQueueManagerConfigure( parms, null );
 ...
 qmConfig.deleteQueueManagerDefinition();
 qmConfig.close();
}
catch (Exception e)
{ ... }
```

MQeQueueManagerConfigure deleteStandardQMDefinitions

構文

```
public void deleteStandardQMDefinitions ( ) throws Exception
```

説明 これは、レジストリーから標準デフォルトのキューの定義とキュー・マネージャーそのものを削除します。定義が存在していなければ、エラーは発生しません。

このメソッドは使い勝手が良く、以下のメソッドと同等の機能があります。

```
deleteDeadLetterQueueDefinition();
deleteSystemQueueDefinition();
deleteAdminQueueDefinition();
deleteAdminReplyQueueDefinition();
deleteQueueManagerDefinition();
```

パラメーター

なし

戻り値 なし

例外

MQeException	MQeQueueManagerConfigure オブジェクトがアクティブにされていない場合、またはレジストリー項目の削除でエラーが生じる場合に出されます。

例

```
try
{
 MQeQueueManagerConfigure qmConfig;
 MQeFields parms = new MQeFields();
 // initialize the parameters
 ...
 qmConfig = new MQeQueueManagerConfigure( parms, null );
 qmConfig.deleteStandardQMDefinitions();
 qmConfig.close();
}
catch (Exception e)
{ ... }
```

MQeQueueManagerConfigure deleteSystemQueueDefinition

構文

```
public void deleteSystemQueueDefinition ( ) throws Exception
```

説明 これは、キュー・マネージャーのレジストリーから SYSTEM.DEFAULT.LOCAL.QUEUE というデフォルトのローカル・キューの定義を削除します。定義が存在していなければ、エラーは発生しません。キューそのものが削除されることはありません。

キューがレジストリーに定義されていなければ、キューにアクセスできません。定義は、**defineDefaultSystemQueue()** を使って再作成することができます。

パラメーター
なし

戻り値 なし

例外

MQeException	MQeQueueManagerConfigure オブジェクトがアクティブにされていない場合、またはレジストリー項目の削除でエラーが生じる場合に示されます。

例

```
try
 MQeQueueManagerConfigure qmConfig;
 MQeFields parms = new MQeFields();
 // initialize the parameters
 ...
 qmConfig = new MQeQueueManagerConfigure( parms, null );
 qmConfig.deleteSystemQueueDefinition();
 ...
 qmConfig.close();
}
catch (Exception e)
{ ... }
```

MQeQueueManagerConfigure queueManagerExists

構文

```
public boolean queueManagerExists( ) throws Exception
```

説明 このメソッドは、レジストリーにキュー・マネージャーの定義があるかどうかを調べます。

パラメーター
なし

戻り値

true レジストリーにキュー・マネージャーの定義がある場合

false レジストリーにキュー・マネージャーの定義がない場合

例外

MQeException MQeQueueManagerConfigure オブジェクトがアクティブにされていない場合、またはレジストリーの読み取りでエラーが生じる場合に出されます。

例

```
try
{
 MQeQueueManagerConfigure qmConfig;
 MQeFields parms = new MQeFields();
 // initialize the parameters
 ...
 qmConfig = new MQeQueueManagerConfigure( parms, null );
 if ( queueManagerExists() )
 {
  ...
 }
 qmConfig.close();
}
catch (Exception e)
{ ... }
```

MQeQueueManagerConfigure setChannelTimeout

構文

```
public void setChannelTimeout( long ChnlTimeout )
```

説明 これは、キュー・マネージャーのチャネル・タイムアウト値を設定します。

このメソッドは、 **defineQueueManager()** の前に呼び出す必要があります。後に呼び出そうとしても、無視されます。

パラメーター

ChnlTimeout チャネル・タイムアウト値 (ミリ秒単位)。

戻り値 なし

例外 なし

例

```
try
{
 MQeQueueManagerConfigure qmConfig;
 MQeFields parms = new MQeFields();
 // initialize the parameters
 ...
 qmConfig = new MQeQueueManagerConfigure( parms, "qmName" + File.separator +
                                                 "Queues" + File.separator );
```

```
 qmConfig.setChannelTimeout( 3600 * 1000 );
 qmConfig.defineQueueManager();
 qmConfig.close();
}
catch (Exception e)
{ ... }
```

MQeQueueManagerConfigure setChnlAttributeRuleName

構文

```
public void setChnlAttributeRuleName( String ChnlAttrRuleName ) throws MQeException
```

説明 このメソッドは、キュー・マネージャーのチャネル属性ルール・クラスの名前を設定します。

このメソッドは、**defineQueueManager()** の前に呼び出す必要があります。後に呼び出そうとしても、無視されます。

パラメーター

ChnlAttrRuleName
チャネル属性ルール・クラスの名前。

戻り値 なし

例外

MQeException 名前が無効な場合に示されます。

例

```
try
{
 MQeQueueManagerConfigure qmConfig;
 MQeFields parms = new MQeFields();
 // initialize the parameters
 ...
 qmConfig = new MQeQueueManagerConfigure( parms, "qmName" + File.separator +
                                                 "Queues" + File.separator );
 qmConfig. setChnlAttributeRuleName( "Examples.Rules.AttributeRule" );
 qmConfig.defineQueueManager();
 qmConfig.close();
}
catch (Exception e)
{ ... }
```

MQeQueueManagerConfigure setDescription

構文

```
public void setDescription (String description )
```

説明 このメソッドは、キュー・マネージャーの説明を設定します。

このメソッドは、 **defineQueueManager()** の前に呼び出す必要があります。後に呼び出そうとしても、無視されます。

パラメーター

description 新しい説明

戻り値 なし

例外 なし

例

```
try
{
 MQeQueueManagerConfigure qmConfig;
 MQeFields parms = new MQeFields();
 // initialize the parameters
 ...
 qmConfig = new MQeQueueManagerConfigure( parms, "qmName" + File.separator +
                                                 "Queues" + File.separator );
 qmConfig.setDescription( "queue manager for " + qmName );
 qmConfig.defineQueueManager();
 qmConfig.close();
}
catch (Exception e)
{ ... }
```

MQeQueueManagerRule

このクラスには、キュー・マネージャーが特定の操作を実行するときに呼び出されるメソッドが含まれています。このような操作の結果は、ルールの影響を受けることがあります。このクラスには、デフォルトのキュー・マネージャーのルールがあります。通常、このようなデフォルトのルールは、所定の MQSeries Everyplace ソリューションに適した振る舞いができるようにオーバーライドされます。

パッケージ

com.ibm.mqe

メソッド

メソッド	目的
activateQueues	このルールは、キュー・マネージャーの起動時に、特定のキューをアクティブにするかどうかを決定します。アクティブにすることのできるキューは、リモート非同期キュー定義、ホーム・サーバー・キュー、およびストア・アンド・フォワード・キューです。
addQueue	このルールは、キュー・マネージャーに新しいキューが追加されたときに呼び出されます。
deleteMessage	このルールは、メッセージの削除操作が行われるときに呼び出されます。
getMessage	このルールは、メッセージの取得操作が行われるときに呼び出されます。
getRetryCount	このルールは、失敗したネットワーク操作での再試行回数を戻します。
peerConnection	このルールは、キュー・マネージャーのピア・チャネル・リスナーが着信接続要求を受信するときに呼び出されます。
putMessage	このルールは、メッセージの書き込み操作が行われるときに呼び出されます。
queueManagerActivate	このルールは、キュー・マネージャーがアクティブにされるときに呼び出されます。
queueManagerClose	このルールは、キュー・マネージャーがクローズされるときに呼び出されます。
removeQueue	このルールは、キュー・マネージャーからキューを削除するときに呼び出されます。
transmit	このルールは、保留メッセージの伝送が行なわれるときに、リモート非同期キュー定義ごとに呼び出されます。このルールがあるので、キューごとに伝送されることを防げます。
triggerTransmission	このルールは、(ここでは) リモート非同期キュー定義に格納された保留メッセージを伝送できるかどうかを示すブール値を戻します。

メソッド	目的
undo	このルールは、取り消し操作が行なわれるときに呼び出されます。

MQeQueueManagerRule activateQueues

構文

```
public boolean activateQueues()
```

説明 このルールは、キュー・マネージャーの起動時に、特定のキューをアクティブにするかどうかを決定します。アクティブにすることのできるキューは、リモート非同期キュー定義、ホーム・サーバー・キュー、およびストア・アンド・フォワード・キューです。

これらのキューをアクティブにすると、キューにあるすべてのメッセージの伝送が試行されることになります。キューに対して操作を実行しない限り、通常はキューはアクティブにされません。キューには、伝送タイマー・スレッドやキューに関連付けられた他の機能がある可能性があるので、キュー・マネージャーを起動したらすぐにキューをアクティブにすると良い場合があります。

パラメーター

なし

戻り値 キュー・マネージャーの起動時に、特定のキューをアクティブにするかどうかを決定するブール値。キュー・マネージャーは、戻された値に基づいて動作します。

例外 なし

例

```
class exampleRules extends MQeQueueManagerRule
{
  ...
  /* cheap rate transmission period start and end times         */
  protected int cheapRatePeriodStart = 18;    /* 18:00 hrs */
  protected int cheapRatePeriodEnd = 9;    /* 09:00 hrs */

  public boolean activateQueues()
  {
    super.activateQueues();
    if ( timeToTransmit() ) /* if OK to transmit */
      return true; /* then activate queues */
    else /* otherwise*/
      return false; /* don't activate queues */
  }

  /* This method determines if the current time is inside the defined    */
  /* cheap rate period of transmission                                       */
  protected boolean timeToTransmit()
  {
    /* get current time */
    long currentTimeLong = System.currentTimeMillis();

    Date date = new Date( currentTimeLong );
    Calendar calendar = Calendar.getInstance();
```

```
    calendar.setTime( date );

    /* get hour */
    int hour = calendar.get( Calendar.HOUR_OF_DAY );

    if ( hour >= cheapRatePeriodStart || hour < cheapRatePeriodEnd )
      return true; /* cheap rate */
    else
      return false; /* not cheap rate */
  }
  ...
}
```

MQeQueueManagerRule addQueue

構文

```
public void addQueue( MQeQueue queue ) throws Exception
```

説明 このルールは、キュー・マネージャーにキューが追加されたときに呼び出されます。ルールはキューの追加前に呼び出されるので、例外を出すことにより、操作を拒否することができます。

またこのルールは、キューをすぐにアクティブにするか、あるいはキューの最初の使用時にアクティブにするかを決定します。このルールのデフォルトの振る舞いでは、すべてのホーム・サーバー・キューがすぐにアクティブにされます。他のすべてのタイプのキューは、最初の使用時にアクティブにされます。

パラメーター

queue キュー・マネージャーに追加される MQeQueue オブジェクト。

戻り値 キューをすぐにアクティブにするか、あるいは最初の使用時にアクティブにするかを示すブール値。

true すぐにキューをアクティブにします

false 最初の使用時にキューをアクティブにします

例外 なし

例

```
class exampleRules extends MQeQueueManagerRule
{
  ...
  /* Don't allow asynchronous queues to be added to this Queue Manager */
  public boolean addQueue( MQeQueue queue ) throws Exception
  {
    boolean result = super.addQueue( queue );
    int accessMode = queue.getAccessMode();
    if ( accessMode == MQeQueue.Queue_ASynchronous )
      throw new MQeException( Except_Rule, "No Asynch Queues" );
    return ( result );
  }
  ...
}
```

関連する関数

removeQueue()

MQeQueueManagerRule deleteMessage

構文

```
public void deleteMessage( String destQMgrName,
                           String destQName,
                           MQeFields filter ) throws Exception
```

説明　このルールは、メッセージの削除操作が行われるときに呼び出されます。ルールは操作が行なわれる前に呼び出されるので、例外を出すことにより、操作を停止することができます。

パラメーター

destQMgrName　この操作の対象となるキューを所有するキュー・マネージャーの名前を示すストリング。ヌルの値は、ローカル・キュー・マネージャーが使われることを示すものです。

destQName　この操作の対象となるキューの名前を示すストリング。

filter　メッセージの削除操作で使うフィルター。これは、メッセージ・フィールド (たとえば、優先順位やメッセージ ID) を含む MQeFields オブジェクトです。

戻り値　なし

例外　なし

例

```
class exampleRules extends MQeQueueManagerRule
{
  ...
  /* This rule blocks message deletes on 'TopSecretQueue' */
  public void deleteMessage( String destQMgr, String destQ, MQeFields filter )
  {
    super.deleteMessage( destQMgr, destQ, filter );
    if( destQMgr == null || destQMgr.equals( Owner.GetName() )
    {
      if ( destQ.equals( "TopSecretQueue" )
        throw new MQeException( Except_Rule, "Can't delete on this Queue" );
    }
  }
  ...
}
```

MQeQueueManagerRule getMessage

構文

```
public void getMessage( String destQMgrName,
                        String destQName,
                        MQeFields filter,
                        MQeAttribute attribute,
                        long confirmId ) throws Exception
```

説明 このルールは、メッセージの取得操作が行われるときに呼び出されます。ルールは操作が行なわれる前に呼び出されるので、例外を出すことにより、操作を停止することができます。

パラメーター

destQMgrName この操作の対象となるキューを所有するキュー・マネージャーの名前を示すストリング。ヌルの値は、ローカル・キュー・マネージャーが使われることを示すものです。

destQName この操作の対象となるキューの名前を示すストリング。

filter メッセージの取得操作で使うフィルター。これは、メッセージ・フィールド (たとえば、優先順位やメッセージ ID) を含む MQeFields オブジェクトです。

attribute メッセージ・レベルのセキュリティーを提供するために使われる MQe Attribute オブジェクト。

confirmId 確実なメッセージ送達を使うかどうかを示す long 値。非ゼロ値であれば、メッセージはターゲット・キューでロックされたままになり、その後に確認のフローが出てくるまで削除されません。ゼロの値であれば、その後で確認のフローがなくても、メッセージがターゲット・キューから削除されます。

この値は、メッセージの取得に失敗するときにも使われます。アプリケーションは、この値を格納しておき、この取得に一致するメッセージを (**undo()** コマンドにより) 前の状態にリセットするときに使用します。

戻り値 なし

例外 なし

例

```
class exampleRules extends MQeQueueManagerRule
{
  ...
  /* This rule only allows GETs from 'OutboundQueue', if a password is  */
  /* supplied as part of the filter                                     */
  public void getMessage( String destQMgr, String destQ, MQeFields filter,
                          MQeAttribute attr, long confirmId )
  {
    super.getMessage( destQMgr, destQ, filter, attr, confirmId );
    if ( destQMgr.equals( Owner.GetName() && destQ.equals( "OutboundQueue" ) )
    {
      if ( !(filter.Contains( "Password" ) )
        throw new MQeException( Except_Rule, "Password not supplied" );
      else
      {
        String pwd = filter.getAscii( "Password" );
        if ( !(pwd.equals( "1234" )) )
          throw new MQeException( Except_Rule, "Incorrect password" );
      }
```

```
    }
  }
  ...
}
```

関連する関数

putMessage()

MQeQueueManagerRule getRetryCount

構文

```
public int getRetryCount()
```

説明 このルールは、ネットワーク操作の試行回数を戻します。キュー・マネージャーは、新しいチャネル・オブジェクトを作成するときに、このルールを呼び出します。このルールで戻される値はチャネルに渡され、ネットワーク操作に失敗するときに使われます。

パラメーター

なし

戻り値 ネットワーク操作の試行回数を示す整数値。キュー・マネージャーは、戻された値に基づいて動作します。

例外 なし

例

```
class exampleRules extends MQeQueueManagerRule
{
  ...
  public int getRetryCount()
  {
    return (2); /* retry a network operation twice */
  }
  ...
}
```

MQeQueueManagerRule peerConnection

構文

```
public void peerConnection( String qmgrName )
```

説明 このメソッドは、キュー・マネージャーのピア・リスナーが別の MQSeries Everyplace キュー・マネージャーからの着信接続要求を検出したときに呼び出されます。接続は、 MQePeerChannel またはその下位クラス経由で行う必要があります。

このルールは、例外を出すことにより、接続の試行をブロックすることができます。

パラメーター

qmgrName 接続を要求している MQSeries Everyplace キュー・マネージャーの名前を示すストリング。

戻り値 なし

例外 なし

例

```
class exampleRules extends MQeQueueManagerRule
{
  ...
  public void peerConnection( String qmgrName )
  {
    /* block any connection attempt from 'RogueQMgr' */
    if ( qmgrName.equals( "RogueQMgr" ) )
      throw new MQeException( Except_Rule, "Connection not allowed" );
  }
  ...
}
```

MQeQueueManagerRule putMessage

構文

```
public void putMessage( String destQMgrName,
                        String destQName,
                        MQeMsgObject msg,
                        MQeAttribute attribute,
                        long confirmId ) throws Exception
```

説明 このルールは、メッセージの書き込み操作が行われるときに呼び出されます。ルールは操作が行なわれる前に呼び出されるので、例外を出すことにより、操作を停止することができます。

パラメーター

destQMgrName この操作の対象となるキューを所有するキュー・マネージャーの名前を示すストリング。ヌルの値は、ローカル・キュー・マネージャーが使われることを示すものです。

destQName この操作の対象となるキューの名前を示すストリング。

msg ターゲット・キューに入れられるメッセージ・オブジェクト

attribute ヌル、またはこのメッセージに関連付けられるオーセンティケーター、暗号機能、圧縮機能を定義する MQeAttribute オブジェクト。

confirmId 確実なメッセージ送達を使うかどうかを示す long 値。非ゼロ値であれば、メッセージはターゲット・キューでロックされ、その後に確認のフローが出てくるまで確認できません。ゼロの値であれば、その後で確認のフローがなくても、メッセージは伝送されます。

さらに、この値は、メッセージの書き込みに失敗するときにも使うことができます。この値を undo コマンドに渡すことにより、アプリケーションは、失敗した書き込み操作によって不完全な状態になっているメッセージがあれば、それらを削除することができます。

戻り値 なし

例外 なし

例

```
class exampleRules extends MQeQueueManagerRule
{
  ...
  /* Only allow msgs containing an ID field to be placed on the Queue */
  public void putMessage( String destQMgr, String destQ, MQeMsgObject msg,
                          MQeAttribute attribute, long confirmId )
  {
    if ( !(msg.Contains( MQe.Msg_MsgId )) )
      throw new MQeException( Except_Rule, "Msg must contain an ID" );
  }
  ...
}
```

関連する関数

getMessage()

MQeQueueManagerRule queueManagerActivate

構文

```
MQeQueueManagerRule queueManagerActivate
```

説明 このルールは、キュー・マネージャーがアクティブにされるときに呼び出されます。

パラメーター

なし

戻り値 なし

例外 なし

例

```
class exampleRules extends MQeQueueManagerRule
{
  ...
  /* default interval between triggers is 60 minutes                    */
  protected int       triggerInterval = 360000;
  /* background thread reference   */
  protected Thread    th              = null;
  /* Called when the Queue manager is activated                         */
  public void queueManagerActivate( ) throws Exception
  {
    super.queueManagerActivate();
    /* background thread which triggers transmission                   */
```

```
    th = new Thread( this, "TriggerThread" );
    th.start();                     /* start timer thread */
  }

  /* Called when a Queue manager Close is called                          */
  public void queueManagerClose( ) throws Exception
  {
    super.QueueManagerClose();
    th.stop();                      /* stop background thread on QMgr close*/
  }

  /* Background thread run method                                         */
  /* Triggers transmission every interval until thread is stopped         */
  public void run()
  {
    try
    {
      while ( true )
      {                             /* sleep for specified interval       */
        Thread.sleep( triggerInterval );
        /* check if ok to transmit                                        */
        if ( triggerTransmission( 0, null ) )
          /* trigger transmission on QMgr (which is rule owner)           */
          ((MQeQueueManager)Owner).triggerTransmission();
      }
    } catch ( Exception e )
    {
      e.printStackTrace( System.err );
    }
  }
  ...
}
```

関連する関数

queueManagerClose()

MQeQueueManagerRule queueManagerClose

構文

```
public void queueManagerClose( ) throws Exception
```

説明 このルールは、キュー・マネージャーをクローズするときに呼び出されます。

パラメーター

なし

戻り値 なし

例外 なし

例

```
class exampleRules extends MQeQueueManagerRule
{
  ...
  /* default interval between triggers is 60 minutes                      */
  protected int       triggerInterval = 360000;
  /* background thread reference   */
  protected Thread    th              = null;
  /* Called when the Queue manager is activated                           */
  public void queueManagerActivate( ) throws Exception
  {
    super.queueManagerActivate();
```

```
    /* background thread which triggers transmission                    */
    th = new Thread( this, "TriggerThread" );
    th.start();                    /* start timer thread */
  }

  /* Called when a Queue manager Close is called                        */
  public void queueManagerClose( ) throws Exception
  {
    super.queueManagerClose();
    th.stop();                     /* stop background thread on QMgr close*/
  }

  /* Background thread run method                                       */
  /* Triggers transmission every interval until thread is stopped       */
  public void run()
  {
    try
    {
      while ( true )
      {                            /* sleep for specified interval      */
        Thread.sleep( triggerInterval );
        /* check if ok to transmit                                      */
        if ( triggerTransmission( 0, null ) )
          /* trigger transmission on QMgr (which is rule owner)         */
          ((MQeQueueManager)Owner).triggerTransmission();
      }
    } catch ( Exception e )
    {
      e.printStackTrace( System.err );
    }
  }
  ...
}
```

関連する関数

queueManagerActivate()

MQeQueueManagerRule removeQueue

構文

```
public void removeQueue( MQeQueue queue ) throws Exception
```

説明 このルールは、キュー・マネージャーからキューを削除するときに呼び出されます。ルールはキューの削除前に呼び出されるので、例外を出すことにより、操作を拒否することができます。

パラメーター

queue キュー・マネージャーから削除される MQeQueue オブジェクト

戻り値 なし

例外 なし

例

```
class exampleRules extends MQeQueueManagerRule
{
  ...
  /* This rule prevents the removal of the Payroll Queue */
```

```
  public void removeQueue( MQeQueue queue ) throws Exception
  {
    if ( queue.getQueueName().equals( "PayrollQueue" ) )
      throw new MQeException( Except_Rule, "Can't delete this queue" );
  }
  ...
}
```

関連する関数

addQueue()

MQeQueueManagerRule transmit

構文

```
public boolean transmit( MQeQueue queue )
```

説明 キュー・マネージャーがすべての保留メッセージを送信しようとすると、伝送を待機しているメッセージを含むキューごとに、このルールが呼び出されます。このルールは、指定したキューのメッセージを伝送できるかどうかを決定します。

パラメーター

queue 伝送を待機しているメッセージがある MQeQueue オブジェクト。

戻り値 このキューに保留されているメッセージを伝送できるかどうかを示すブール値。キュー・マネージャーは、戻された値に基づいて動作します。

例外 なし

例

```
class exampleRules extends MQeQueueManagerRule
{
  ...
  /* cheap rate transmission period start and end times                    */
  protected int cheapRatePeriodStart = 18;          /* 18:00 hrs         */
  protected int cheapRatePeriodEnd = 9;            /* 09:00 hrs         */

  /* This rule allows queue transmission if current time is during the     */
  /* cheap rate transmission period                                        */
  /* If the current time is not during the cheap rate transmission period  */
  /* then transmission is only allowed if the queue is high priority       */
  public boolean transmit( MQeQueue queue )
  {
    if ( timeToTransmit() )
      return true;                              /* cheap rate          */
    else
      if ( queue.GetPriority() > 4 )
        return true;                                /* high priority Q     */
  }

  /* This method determines if the current time is inside the defined    */
  /* cheap rate period of transmission                                     */
  protected boolean timeToTransmit()
  {
    /* get current time */
    long currentTimeLong = System.currentTimeMillis();
```

```
    Date date = new Date( currentTimeLong );
    Calendar calendar = Calendar.getInstance();
    calendar.setTime( date );
    /* get hour */
    int hour = calendar.get( Calendar.HOUR_OF_DAY );

    if ( hour >= cheapRatePeriodStart || hour < cheapRatePeriodEnd )
      return true; /* cheap rate */
    else
      return false; /* not cheap rate */
  }
  ...
}
```

関連する関数

triggerTransmission()

MQeQueueManagerRule triggerTransmission

構文

```
public boolean triggerTransmission( int noofMsgs, MQeFields msgFields )
```

説明 このメソッドは、キュー・マネージャー内のリモート非同期キュー定義に格納されている保留メッセージの伝送を許可します。

このルールは、以下の 2 つの場合に、キュー・マネージャーによって呼び出されます。

- キュー・マネージャーが、(**MQeQueueManager.triggerTransmission()** メソッドを使用して) 保留メッセージすべてを伝送するよう指示される場合。
- メッセージが、非同期と定義されているリモート・キューへ送信される場合。キュー・マネージャーはこのルールを呼び出し、すべての保留メッセージを伝送するかどうか確認します。

パラメーター

noofMsgs リモート非同期キューで伝送を待機しているメッセージの数

msgFields キュー・マネージャーは、(**MQeQueueManager.triggerTransmission()** メソッドを使用して)、すべての保留メッセージを伝送するよう指示されているので、このルールが呼び出される場合、このパラメーターは、ヌルです。

リモート非同期キュー定義のメッセージが到着して、このルールが呼び出されると、このパラメーターは、新しく到着したメッセージに含まれる特定のフィールドを含む MQeFields オブジェクトになります。

パラメーターに示されるフィールドは、以下のとおりです。

- Message UID (メッセージ UID) (起点キュー・マネージャー + タイム・スタンプ)

- Message ID (メッセージ ID) (元のメッセージに含まれていれば)
- Correlation ID (相関 ID) (元のメッセージに含まれていれば)
- Priority (優先順位) (元のメッセージに含まれていれば)

戻り値 ルールが保留メッセージの伝送を今回許可するかどうかを示すブール値。キュー・マネージャーは、戻された値に基づいて動作します。

例外 なし

例

```
class exampleRules extends MQeQueueManagerRule
{
  ...
  /* default interval between triggers is 60 minutes                        */
  protected int       triggerInterval = 360000;
  /* background thread reference   */
  protected Thread    th              = null;
  /* Called when the Queue manager is activated                             */
  public void queueManagerActivate( ) throws Exception
  {
    super.queueManagerActivate();
    /* background thread which triggers transmission                       */
    th = new Thread( this, "TriggerThread" );
    th.start();                     /* start timer thread */
  }

  /* Called when a Queue manager Close is called                            */
  public void queueManagerClose( ) throws Exception
  {
    super.queueManagerClose();
    th.stop();                      /* stop background thread on QMgr close*/
  }

  /* Background thread run method                                           */
  /* Triggers transmission every interval until thread is stopped           */
  public void run()
  {
    try
    {
      while ( true )
      {                             /* sleep for specified interval        */
        Thread.sleep( triggerInterval );
        /* check if ok to transmit                                          */
        if ( triggerTransmission( 0, null ) )
          /* trigger transmission on QMgr (which is rule owner)             /
          ((MQeQueueManager)Owner).triggerTransmission();
      }
    } catch ( Exception e )
    {
      e.printStackTrace( System.err );
    }
  }

  /* Decides if transmission of messages is allowed */
  public boolean triggerTransmission( int noOfMsgs, MQeFields msgFields )
  {
    return true; /* always allow transmission */
  }
  ...
}
```

関連する関数

transmit()

MQeQueueManagerRule undo

構文

```
public void undo(String destQMgrName,
                 String destQName,
                 long confirmId ) throws Exception
```

説明 このルールは、メッセージのやり直し操作が行われるときに呼び出されます。ルールは操作が行なわれる前に呼び出されるので、例外を出すことにより、操作を停止することができます。

パラメーター

destQMgrName この操作の対象となるキューを所有するキュー・マネージャーの名前を示すストリング。ヌルの値は、ローカル・キュー・マネージャーが使われることを示すものです。

destQName この操作の対象となるキューの名前を示すストリング。

confirmId やり直しする予定の操作で使う confirm Id (確認 ID) を示す long 値。この confirm Id に一致するメッセージはすべて、それぞれの前の状態に復元されます。

戻り値 なし

例外 なし

例

```
class exampleRules extends MQeQueueManagerRule
{
  ...
  public void undo( String destQMgr, String destQ, long confirmId )
  {
    /* log the undo event */
    log( MQe_Log_Warning, Event_QueueManager_Undo,
         destQMgr + "+" + destQ );
  }
  ...
}
```

MQeQueueRule

キュー・ルールは、MQSeries Everyplace キューの動作を制御します。キュー・ルールは、キューそのものがアクティブになるときに、そのキューによってアクティブにされます。キューの操作時には、ルールは、特定のイベントが生じるとき (たとえば、メッセージが書き込まれたとき、メッセージが満了するとき、重複したメッセージが到着するときなど) に呼び出されます。そして、ルールは、キューがこのようなイベントを処理するときの方法を決定します。

キュー・ルールの基本的な設定はこのクラスで定義されますが、ソリューションに応じてキューの動作を変更する場合に、設定を拡張するようにします。

キューは、メッセージをキュー・ストアに入れておきます。これは、一般的に、ディスク・ドライブのような永続的なタイプのストレージですが、必ずしもそうでなければならないわけではありません。キューには、キュー・ストアに入れられるメッセージごとに、索引項目があります。この索引項目は、メッセージをロックするかしないかなど、メッセージの状態情報で構成されています。さらに、索引項目には、特定のメッセージ・フィールドがあり、これらを索引フィールドといいます。デフォルトの索引フィールドは、「メッセージ固有 ID (message unique ID)」、「メッセージ ID (message ID)」、「相関 ID (correlation ID)」、および「メッセージ優先順位 (message priority)」です。これらのフィールドはほとんどのメッセージに存在するものであり、フィールドをメモリーに格納しておけば、メッセージ検索が速いので、これらのフィールドは格納されます。

indexEntry() ルールは、索引項目が作成されるときに必ず呼び出されます。このことは、新しいメッセージがキューに書き込まれるとき、あるいは、キューがアクティブになり、以前のセッションでキュー・ストアに残されたメッセージを読み取るときに行なわれます。ルールでは、ソリューションに合わせて作成時の索引項目を変更し、別のフィールド (複数可) を索引に追加することができるので、メッセージの検索時間を短くできるようになります。

キューは使用回数を管理しています。この数は、キューがアクティブになると増加し、キューをクローズすると減少します。また、リモート・キュー・マネージャーがキューに接続すると、使用回数は増加します。この接続を確立するときに使われたチャネルおよび移送機能が破棄されると、使用回数は減少します。使用回数が変化するたびに、**useCountChanged()** ルールが呼び出されます。

キューに入れられているメッセージは、オーセンティケーターおよび暗号機能によって保護することができます。さらに、圧縮機能を使ってメッセージを圧縮することもできます。オーセンティケーター、暗号機能、および圧縮機能はともに、キューの属性として知られており、キューに関連付けられる適切な MQeAttribute オブジェクトを指定することによって定義されます。キューの属性を置き換えようとすると、**attributeChange()** ルールが必ず呼び出されます。

注: 別の属性を使って格納したメッセージがキューにあるときに、キューの属性を変更する場合、その新しい属性ではメッセージをリカバリーできない可能性があるため、メッセージが失われてしまう場合があります。

メッセージが、キューの有効期限間隔が切れてもキューに残っている場合、または、メッセージそれ自体の有効期限間隔が切れている場合、 messageExpired() ルールが呼び出されます。そしてこのルールは、メッセージに対する処置を決定しますが、一般には、メッセージは削除されるか、送達不能キューに入れられます。

パッケージ

com.ibm.mqe

メソッド

メソッド	目的
addListener	このルールは、メッセージ・リスナーがキューに追加されると呼び出されます。
attributeChange	このルールは、キューの属性を変更しようとすると呼び出されます。この属性では、キューで使われるオーセンティケーター、暗号機能、および圧縮機能が定義されます。メッセージは、定義された属性に基づいて格納されます。
browseMessage	このルールは、メッセージを、ブラウズ要求を発行したアプリケーションに戻されるメッセージ群に含めるかどうかを決定します。
deleteMessage	このルールは、メッセージの削除操作が行われるときに呼び出されます。
duplicateMessage	このルールは、重複したメッセージがキューに書き込まれる場合に呼び出されます。
filterMessage	このルールは、**getMessage**、**browseMessage** または **deleteMessage** メソッドの呼び出し中に、メッセージ・フィルターが適用される場合に呼び出されます。
getMessage	このルールは、メッセージがメッセージの取得要求を満たすものであることが分かった場合に呼び出されます。このルールでは、メッセージがメッセージの取得要求を満たせるかどうかを決定できます。
getPendingMessage	このルールは、保留メッセージの取得要求を受け取ると呼び出されます。
indexEntry	このルールは、索引項目が作成されると必ず呼び出されます。
messageExpired	このルールは、メッセージがキューの有効期限間隔を超過したか、メッセージ自体の有効期限間隔が切れた場合に呼び出されます。その後、このルールによって、メッセージを満了するかどうか、満了している場合は、メッセージをどのように処理するかが決まります。

メソッド	目的
putMessage	このルールは、メッセージの書き込み要求が行なわれるときに呼び出されます。
queueActivate	このルールは、キューがアクティブにされるときに呼び出されます。
queueClosed	このルールは、キューがクローズされるときに呼び出されます。
removeListener	このルールは、メッセージ・リスナーがキューから除去されるときに呼び出されます。
resetMessageLock	このルールは、メッセージに対するロック状態をリセットする要求が出されるときに呼び出されます。メッセージは、ロックが解除された状態にリセットされます。この関数を実行できるのは、MQSeries 管理者だけです。ルールは、例外を出すことにより、リセットの発生を防ぐことができます。
undo	このルールは、やり直し操作が行なわれるときに呼び出され、メッセージをやり直し操作で処理されるメッセージ群に含めるかどうかを決定します。
useCountChanged	このルールは、使用回数が変化するたびに呼び出されます。

MQeQueueRule addListener

構文

```
public void addListener( MQeMessageListenerInterface listener,
                         MQeFields filter) throws Exception
```

説明 このルールは、メッセージ・リスナーがキューに追加されると呼び出されます。ルールは、例外を出すことにより、リスナーの追加要求を拒否できます。

パラメーター

listener MQSeries Everyplace メッセージ・イベントに加入しているオブジェクトへの参照。このオブジェクトは、MQeMessageListenerInterface を実装する必要があります。

filter ヌル、または メッセージ・フィールドを含む MQeFields オブジェクト。値がヌルの場合、リスナーは、キューにあるすべてのメッセージのイベントを受信します。

メッセージ・フィールドを含む MQeFields オブジェクトを指定する場合、リスナーは、フィルターに含まれるフィールドと一致するフィールドを持つメッセージに関係したイベントだけを受信します。

戻り値 なし

例外 なし

例

```
class exampleQueueRules extends MQeQueueRule
{
  ...
  /* This rule logs the addition of a message listener */
  public void addListener( MQeMessageListenerInterface listener,
                           MQeFields filter ) throws Exception
  {
    log( MQe_Log_Information, Event_Queue_AddMsgListener,
         "Added listener on queue " +
         ((MQeQueue)owner).getQueueManagerName() + "+" +
         ((MQeQueue)owner).getQueueName() );
  }

  public void queueActivate()
  { /* create a new log file */
    try
    {
      logFile = new LogToDiskFile( "¥log.txt" );
    }
    catch( Exception e )
    {
      e.printStackTrace( System.err );
    }
  }

  public void queueClose()
  { /* close log file */
    logFile.close();
  }
  ...
}
```

関連する関数

- **removeListener()**

MQeQueueRule attributeChange

構文

```
public void attributeChange( MQeAttribute attribute ) throws Exception
```

説明 このメソッドは、キューの属性を変更しようとすると呼び出されます。この属性では、キューで使われるオーセンティケーター、暗号機能、および圧縮機能が定義されます。メッセージはすべて、このキューの属性を使って格納されます。

ルールは、例外を出すことにより、属性の変更要求を拒否できます。

パラメーター

attribute ヌル、または変更が許可される場合に、キューで使われるオーセンティケーター、暗号機能および圧縮機能を定義する

MQeAdminQueueAdminMsg オブジェクト。ヌルの場合、キューで使われる属性はありません。

戻り値 なし

例外

例

```
class exampleQueueRules extends MQeQueueRule
{
  ...
  /* This rule only allows the queue's attribute to be changed if it was */
  /* not previously set */
  /* The queue object is the owner of the rule */
  public void attributeChange( MQeAttribute attribute ) throws Exception
  {
    if ( ((MQeQueue) owner).getQueueAttribute() != null )
      throw new MQeException( Except_Rule, "Attribute already set" );
  }
  ...
}
```

MQeQueueRule browseMessage

構文

```
public boolean browseMessage( MQeMsgObject msg,
                              long confirmId ) throws Exception
```

説明 1 回のメッセージのブラウズ操作で、キューに保留されているゼロ個以上のメッセージを突き合わせます。このメソッドは、メッセージが突き止められるたびに呼び出されます。このルールは、メッセージを、ブラウズ要求を発行したアプリケーションに戻されるメッセージ群に含めるかどうかを決定します。

このルールが例外を出すと、ブラウズ操作は終了します。

パラメーター

msg ブラウズするメッセージを含む MQeMsgObject。

confirmId このブラウズ操作で使われる *confirmID* である long 値。*confirmID* は、障害時に、メッセージを復元するときに使用します。

戻り値 メッセージを、ブラウズ要求を発行したアプリケーションに戻されるメッセージ群に含めるかどうかを示すブール値。

True メッセージを含めることができます。

False メッセージを含めることができません。

例外

例

```
class exampleQueueRules extends MQeQueueRule
{
  ...
```

```
  /* This rule only allows messages of type 'OrderResponse' to be browsed */
  public boolean browseMessage( MQeMsgObject msg,
                                long confirmID ) throws Exception
  {
    /* get message type field */
    String msgType = msg.getAscii( "MsgType" );
    /* what message type is it? */
    if ( msgType.equals( "OrderResponse" ) )
      return (true); /* allow browse */
    else
      return (false); /* don't allow browse */
  }
  ...
}
```

MQeQueueRule deleteMessage

構文

```
  public void deleteMessage( MQeFields filter ) throws Exception
```

説明 このルールは、メッセージの削除操作が行われるときに呼び出されます。ルールは、例外を出すことにより、メッセージの削除要求を拒否できます。

パラメーター

filter メッセージのフィルターを含む MQeFields オブジェクト。削除操作を成功させるため、このフィルターには、メッセージの UID を含める必要があります。

戻り値 なし

例外

例

```
class exampleQueueRules extends MQeQueueRule
{
  ...
  /* This rule requires that the filter contain a password  */
  public void deleteMessage( MQeFields filter ) throws Exception
  {
    if ( filter != null && filter.contains( "Password" ) )
    {
      String pswd = filter.getAscii( "Password" );
      if ( pswd.equals( "12345678" ) )
      { /* remove password from filter */
        filter.delete( "Password" );
        return;
      }
      else
        throw new MQeException( Except_Rule, "Incorrect password" );
    }
    throw new MQeException( Except_Rule, "No Password" );
  }
  ...
}
```

MQeQueueRule duplicateMessage

構文

```
public void duplicateMessage( MQeMsgObject msg,
                              long confirmId ) throws Exception
```

説明 このルールは、重複したメッセージがキューに書き込まれる場合に呼び出されます。

パラメーター

msg 重複したメッセージを含む MQeMsgObject。

confirmId この書き込み操作で使われる *confirmID* である long 値。*confirmID* は、障害時に、メッセージを復元するときに使用します。

戻り値 なし

例外

例

```
class exampleQueueRules extends MQeQueueRule
{
  ...
  /* This rule logs the duplicate message exception */
  public void duplicateMessage( MQeMsgObject msg,
                        long confirmID ) throws Exception
  {
    /* get message UID */
    MQeFields msgUID = msg.getMsgUIDFields();
    /* log the duplicate message exception */
    log( MQe_Log_Warning, Event_Queue_Duplicate,
         msgUID.getAscii( MQe.Msg_OriginQMgr ) + " " + " +
         msgUID.getLong( MQe.Msg_Time ) );
  }

  public void queueActivate()
  { /* create a new log file */
    try
    {
      logFile = new LogToDiskFile( "¥log.txt" );
    }
    catch( Exception e )
    {
      e.printStackTrace( System.err );
    }
  }

  public void queueClose()
  { /* close log file */
```

```
    logFile.close();
  }
  ...
}
```

MQeQueueRule filterMessage

構文

```
public boolean filterMessage( int what, MQeFields filter )
```

説明 このルールは、**getMessage()**、 **browseMessage()** または **deleteMessage()** 呼び出し中に、メッセージ・フィルターが適用される場合に呼び出されます。このキュー・ルールは、そのフィルターに特定のフィールドが存在し、なおかつその中に特定の値が含まれているかどうかを確認するために、検査を行います。それから、ルールはこの情報を使用し、操作を許可または拒否します。

パラメーター

what 実行される操作を示す整数。

- 0 = ブラウズ
- 1 = 取得
- 2 = 削除

filter MQeFields オブジェクト。ヌル、またはキューのサブセット・メッセージに操作を制限するのに使用する検索フィルター。

戻り値

true 要求された操作の続行を許可します。

false 要求された操作を拒否します。

例外 なし

例

```
public class ExampleQueueRule extends MQeQueueRule {
...
public boolean filterMessage( int what , MQeFields filter ) {
boolean result = true ; // Allow all operations by default, regardless of filter.
return result ;
}
...
}
```

関連する関数

- **getMessage()**
- **browseMessage()**
- **deleteMessage()**

MQeQueueRule getMessage

構文

```
public boolean getMessage( MQeMsgObject msg,
                           long confirmId ) throws Exception
```

説明 このルールは、メッセージがメッセージの取得要求を満たすものであることが分かった場合に呼び出されます。このルールでは、そのメッセージがメッセージの取得要求を満たせるかどうかを決定できます。

ルールにより、メッセージがメッセージの取得要求を満たせない場合、キューは、要求を満たす別のメッセージを探します。

ルールは、例外を出すことにより、メッセージの取得要求を終了できます。

パラメーター

msg メッセージの取得要求を満たすメッセージを含む MQeMsgObject。

confirmId この取得要求で使われる *confirmID* である long 値。*confirmID* は、障害時に、メッセージを復元するときに使用します。

戻り値 なし

例外 なし

例

```
class exampleQueueRules extends MQeQueueRule
{
  ...
  /* This rule disallows a get request on a message if the message states */
  /* that it can only be browsed */
  public boolean getMessage( MQeMsgObject msg,
                             long confirmId ) throws Exception
  { /* does the message contain a field which states what operations are */
    /* allowed upon it */
    if ( msg.contains( "AllowableOperations" ) )
    {
      String allowedOperations = msg.getAscii( "AllowableOperations" );
      /* if the message states that it can only be browsed disallow the get */
      if ( allowedOperations.equals( "BrowseOnly" ) )
        return (false);
    }
    return (true);
  }
  ...
}
```

関連する関数

- putMessage

MQeQueueRule getPendingMessage

構文

```
public void getPendingMessage( String queueManagerName,
                               MQeFields filter,
                               long confirmId ) throws Exception
```

説明 MQSeries Everyplace キュー・マネージャーは、ホーム・サーバー・キュー経由で、ホーム・サーバーからキュー・マネージャーにあてられたメッセージを収集することができます。ホーム・サーバーは、クライアントあてのメッセージを、1 つ以上のストア・アンド・フォワード・キューに格納します。ホーム・サーバー・キューは、 **MQeStoreAndForwardQueue.getPendingMessage()** メソッドを使い、ホーム・サーバーのストア・アンド・フォワード・キューに接触します。その MQSeries Everyplace キュー・マネージャー向けの保留メッセージがあれば、戻されます。

このルールは、保留メッセージの取得要求を受け取ると呼び出されます。

パラメーター

queueManagerName
保留メッセージの取得要求を開始した MQSeries Everyplace キュー・マネージャーの名前を示すストリング。

filter
ヌル、または保留メッセージを突き止めるときに使うメッセージ・フィルターを含む MQeFields オブジェクト。ヌルの場合、 *queueManagerName* で指定したキュー・マネージャーへあてられた、利用可能な最初のメッセージが戻されます。

confirmId
この操作用の *confirmID* である long 値。 *confirmID* は、障害時に、メッセージを復元するときに使用します。

戻り値 なし

例外

例

```
class exampleQueueRules extends MQeQueueRule
{
  ...
  /* This rule requires that the filter contain a password  */
  /* (For this rule to work correctly in would be necessary to override */
  /* MQeHomeServerQueue so that the message filter sent to the Store & */
  /* Forward Queue was non-null) */
  public void getPendingMessage( String queueManagerName, MQeFields filter,
                                 long confirmId ) throws Exception
  {
    if ( filter != null && filter.contains( "Password" ) )
    {
      String pswd = filter.getAscii( "Password" );
      if ( pswd.equals( "123456878" ) )
      { /* remove password from filter */
        filter.delete( "Password" );
        return;
      }
    }
```

```
    throw new MQeException( Except_Rule, "No Password" );
  }
  ...
}
```

MQeQueueRule indexEntry

構文

```
public void indexEntry( MQeFields entry,
                        MQeMsgObject msg ) throws Exception
```

説明 このルールは、キューが索引項目を作成するときに呼び出されます。このことは、キューに以前のセッションのメッセージが残っている場合で、新しいメッセージがキューに書き込まれるとき、そしてキューがアクティブにされるときに行なわれます。

この索引項目には、メッセージについての状態情報だけでなく、メッセージの検索を速くするために保持されている特定の索引フィールドが含まれています。それらのフィールドは、以下のとおりです。

- MQe Unique ID (MQe 固有 ID) (MQe.Msg_OriginQMgr + MQe.Msg_Time)
- MQ Series Message ID (MQ Series メッセージ ID) (MQe.Msg_ID)
- MQ Series Correlation ID (MQ Series 相関 ID) (MQe.Msg_CorrelID)
- Message Priority (メッセージ優先順位) (MQe.Msg_Priority)

パラメーター

entry 保留メッセージの取得要求を開始した MQSeries Everyplace キュー・マネージャーの名前を示すストリング。

filter ブランクの索引項目を含む MQeFields オブジェクト。必要であれば、ルールによって、フィールドをこのオブジェクトへ追加できます。

msg 索引項目を作成する対象のメッセージを含む MQeMsgObject。

戻り値 なし

例外

例

```
class exampleQueueRules extends MQeQueueRule
{
  ...
  /* if the message contains a customer number field - then add this field */
  /* to the message's index entry.  */
  /* This will enable faster message searching */
  public void indexEntry( MQeFields entry,
                          MQeMsgObject msg ) throws Exception
  {
    if ( msg.contains( "Cust_No" ) )
```

```
        entry.copy( msg, true, "Cust_No" );
    }
    ...
}
```

MQeQueueRule messageExpired

構文

```
public boolean messageExpired( MQeFields entry,
                               MQeMsgObject msg ) throws Exception
```

説明 このルールは、メッセージがキューの有効期限間隔を超過したか、メッセージ自体の有効期限間隔が切れた場合に呼び出されます。メッセージをアクセスするたびに、メッセージが有効期限間隔を超過していないか調べる検査が行われます。

その後、このルールによって、メッセージを満了するかどうか、満了している場合は、その後でメッセージをどのように処理するかが決まります。

パラメーター

entry 満了したメッセージの索引項目を含む MQeFields オブジェクト。

msg 満了したメッセージを含む MQeMsgObject。

戻り値 メッセージが満了したかどうかを示すブール値。

true メッセージが満了しているので削除できます。

false メッセージは満了していません。

例外

例

```
class exampleQueueRules extends MQeQueueRule
{
  ...
  /* This rule puts a copy of any expired messages to a Dead Letter Queue */
  public boolean messageExpired( MQeFields entry,
                                 MQeMsgObject msg ) throws Exception
  {
    /* Get the reference to the Queue Manager */
    MQeQueueManager qmgr = MQeQueueManager.getReference(
                                   ((MQeQueue)owner).getQueueManagerName() );
    /* need to set re-send flag so that put of message to new queue isn't */
    /* rejected /
    msg.putBoolean( MQe.Msg_Resend, true );
    /* if the message contains an expiry interval field - remove it */
    if ( msg.contains( MQe.Msg_ExpireTime )
      msg.delete( MQe.Msg_ExpireTime );
    /* put message onto dead letter queue */
    qmgr.putMessage( null, "DEAD.LETTER.QUEUE", msg, null, 0 );
    /* return true & the message will be deleted from the queue */
    return (true);
  }
  ...
}
```

MQeQueueRule putMessage

構文

```
public void putMessage( MQeMsgObject msg,
                        long confirmID ) throws Exception
```

説明 このルールは、メッセージの書き込み要求が行なわれるときに呼び出されます。ルールは、例外を出すことにより、キューにメッセージが書き込まれることを防ぐことができます。

パラメーター

msg キューに書き込まれるメッセージを含む MQeMsgObject。

confirmID この操作用の *confirmID* を含む long 値。 *confirmID* は、障害時に、ロックされたメッセージを復元するときに使用します。

戻り値 なし

例外

例

```
class exampleQueueRules extends MQeQueueRule
{
  ...
  /* This rule blocks a message Put if the message priority is less than 5 */
  public void putMessage( MQeMsgObject msg, long confirmID ) throws Exception
  {
    if ( (msg.contains( MQe.Msg_Priority )) &&
         (msg.getByte( MQe.Msg_Priority ) < 5) )
      throw new MQeException( Except_Rule, "Priority too low" );
  }
  ...
}
```

関連する関数

- **getMessage**

MQeQueueRule queueActivate

構文

```
public void queueActivate()
```

説明 このルールは、キューがアクティブにされるときに呼び出されます。

パラメーター

なし

戻り値 なし

例外 なし

例

```
class exampleQueueRules extends MQeQueueRule
{
  ...
  /* This rule logs the activation of the queue */
  public void queueActivate()
  {
    try
    {
      logFile = new LogToDiskFile( ¥¥log.txt );
      log( MQe_Log_Information, Event_Queue_Activate, "Queue " +
           ((MQeQueue)owner).getQueueManagerName() + " + " +
           ((MQeQueue)owner).getQueueName() + " active" );
    }
    catch( Exception e )
    {
      e.printStackTrace( System.err );
    }
  }

  public void queueClose()
  { /* close log file */
    logFile.close();
  }
  ...
}
```

関連する関数

- **queueClose()**

MQeQueueRule queueClose

構文

```
public void queueClose()
```

説明 このルールは、キューがクローズされるときに呼び出されます。

パラメーター

なし

戻り値 なし

例外 なし

例

```
class exampleQueueRules extends MQeQueueRule
{
  ...
  /* This rule logs the closure of the queue */
  public void queueClose()
  {
    try
    {
      log( MQe_Log_Information, Event_Queue_Closed, "Queue " +
           ((MQeQueue)owner).getQueueManagerName() + " + " +
```

```
            ((MQeQueue)owner).getQueueName() + " closed" );
          /* close log file */
          logFile.close();
        }
        catch( Exception e )
        {
          e.printStackTrace( System.err );
        }
      }
      public void queueActivate()
      {
        try
        {
          logFile = new LogToDiskFile( ¥¥log.txt );
          log( MQe_Log_Information, Event_Queue_Activate, "Queue " +
               ((MQeQueue)owner).getQueueManagerName() + " " + " +
               ((MQeQueue)owner).getQueueName() + " active" );
        }
        catch( Exception e )
        {
          e.printStackTrace( System.err );
        }
      }  ...
    }
```

関連する関数

- **queueActivate()**

MQeQueueRule removeListener

構文

```
public void removeListener( MQeMessageListenerInterface listener,
                            MQeFields filter ) throws Exception
```

説明 このルールは、メッセージ・リスナーがキューから除去されるときに呼び出されます。ルールは、例外を出すことにより、リスナーが削除されることを防ぐことができます。

パラメーター

listener MQSeries Everyplace メッセージ・イベントに加入しているオブジェクト。このオブジェクトは、MQeMessageListenerInterface を実装する必要があります。

filter ヌル、またはメッセージ・フィルターを含む MQeFields オブジェクト。このフィルターは、このリスナーを作成したリスナーの追加コマンドで使ったフィルターと一致していなければなりません。

戻り値 なし

例外

例

```
class exampleQueueRules extends MQeQueueRule
{
  ...
  /* This rule logs the removal of the message listener */
  public void removeListener( MQeMessageListenerInterface listener,
                              MQeFields filter ) throws Exception
  {
    log( MQe_Log_Information, Event_Queue_RemoveMsgListener,
         "Removed listener on queue " +
           ((MQeQueue)owner).getQueueManagerName() + " + " +
         ((MQeQueue)owner).getQueueName() );
  }

  public void queueActivate()
  { /* create a new log file */
    try
    {
      logFile = new LogToDiskFile( ¥¥log.txt );
    }
    catch( Exception e )
    {
      e.printStackTrace( System.err );
    }
  }
  public void queueClose()
  { /* close log file */
    logFile.close();
  }
  ...
}
```

関連する関数

- **addListener**

MQeQueueRule resetMessageLock

構文

```
public void resetMessageLock( MQeFields filter ) throws Exception
```

説明 このルールは、メッセージに対するロック状態をリセットする要求が出されるときに呼び出されます。メッセージは、ロックが解除された状態にリセットされます。この関数を実行できるのは、MQSeries Everyplace 管理者だけです。ルールは、例外を出すことにより、リセットの発生を防ぐことができます。

パラメーター

filter メッセージのフィルターを含む MQeFields オブジェクト。このフィルターは、ロック状態をリセットするメッセージを突き止めるときに使います。

戻り値 なし

例外

例

```
class exampleQueueRules extends MQeQueueRule
{
  ...
  /* This rule logs the message lock reset */
  public void resetMessageLock( MQeFields filter ) throws Exception
  { /* get message UID */
    if ( filter.contains( MQe.Msg_Time ) &&
         filter.contains( MQe.Msg_OriginQMgr ) )
    {
      String originQMgr = filter.getAscii( MQe.Msg_OriginQMgr );
      long timeStamp    = filter.getLong( MQe.Msg_Time );

      log( MQe_Log_Information, Event_Queue_ResetMessageLock,
           "Message " + originQMgr + ":" + timeStamp + " on queue " +
           ((MQeQueue)owner).getQueueManagerName() + " " + " +
           ((MQeQueue)owner).getQueueName() + " has been reset" );
    }
  }

  public void queueActivate()
  { /* create a new log file */
    try
    {
      logFile = new LogToDiskFile( ¥¥log.txt );
    }
    catch( Exception e )
    {
      e.printStackTrace( System.err );
    }
  }

  public void queueClose()
  { /* close log file */
    logFile.close();
  }

  ...
}
```

MQeQueueRule undo

構文

```
public boolean undo( MQeFields filter ) throws Exception
```

説明 1 回のメッセージのやり直し操作で、キューに入れられている 0 個以上のメッセージを突き止めます。このルールは、突き止められるメッセージごとに呼び出され、メッセージをやり直し操作で処理されるメッセージ群に含めるかどうかを決定します。

ルールは、例外を出すことにより、やり直し操作を終了できます。

パラメーター

filter メッセージのフィルターを含む MQeFields オブジェクト。やり直し操作の対象となるのは、フィルターに一致するメッセージだけです。

confirmId 元に戻す操作で使用する *confirmID* である long 値。この confirm Id に一致するメッセージはすべて、それぞれの前の状態に復元されます。

戻り値

このメッセージをやり直し操作で処理されるメッセージ群に含めるかどうかを示すブール値。

true このメッセージをやり直し操作で処理されるメッセージ群に含めます。

false このメッセージをやり直し操作で処理されるメッセージ群に含めません。

例外

例

```
class exampleQueueRules extends MQeQueueRule
{
  ...
  /* This rule logs the message reset */
  public boolean undo( MQeFields filter ) throws Exception
  { /* get message UID */
    if ( filter.contains( MQe.Msg_Time ) &&
         filter.contains( MQe.Msg_OriginQMgr ) )
    {
      String originQMgr = filter.getAscii( MQe.Msg_OriginQMgr );
      long timeStamp    = filter.getLong( MQe.Msg_Time );

      log( MQe_Log_Information, Event_Queue_ResetMessageLock,
           "Message " + originQMgr + ":" + timeStamp + " on queue " +
           ((MQeQueue)owner).getQueueManagerName() + " + " +
           ((MQeQueue)owner).getQueueName() + " has been reset" );
    }
    return (true);
  }
  public void queueActivate()
  { /* create a new log file */
    try
    {
      logFile = new LogToDiskFile( ¥¥log.txt );
    }
    catch( Exception e )
    {
      e.printStackTrace( System.err );
    }
  }

  public void queueClose()
```

```
  { /* close log file */
    logFile.close();
  }
  ...
}
```

MQeQueueRule useCountChanged

構文

```
public void useCountChanged( int useCount ) throws Exception
```

説明 このルールは、キューの使用回数が変わるときに呼び出されます。使用回数は、キューに接続しているユーザー数を示す計測単位です。使用回数が変わるのは、キューがアクティブになるかクローズするとき、そして、リモート・キュー・マネージャーが、 MQeTransporter を使用してキューに接続するかキューを切断するときです。

パラメーター

useCount キューの現在の使用回数。

戻り値 なし

例外

例

```
class exampleQueueRules extends MQeQueueRule
{
  ...
  /* do not allow the use count to exceed ten */
  public void useCountChanged( int useCount ) throws Exception
  {
    if ( useCount == 10 )
      throw new MQeException( Except_Rule, "Too many users" );
  }
  ...
}
```

MQeRule

これは、すべての MQSeries Everyplace ルール・クラスの基本機能の派生元であるスーパークラスです。

パッケージ

com.ibm.mqe

コンストラクター

MQeRule

構文

```
public MQeRule( )
```

説明 MQeRule オブジェクトを構成します。

パラメーター

なし

戻り値 なし

例外 なし

例

メソッド

メソッド	目的
activate	このメソッドは、コンストラクターまたはクローズ・メソッドを使用した後で、ルールを使用する前に呼び出されます。
close	このメソッドは、ルールを使用停止にする必要があることを示します。
newRule	このメソッドは、所有オブジェクトで現行ルールを別のルールに変更したい場合に、現行ルールが別のルールにより置き換え可能かどうかを調べるために呼び出します。

MQeRule activate

構文

```
public void activate( Object thisOwner )
```

説明 このメソッドは、コンストラクターを使用した後で、ルールを使用する前に呼び出されます。サブクラスにより、このメソッドはオーバーライドされます。

パラメーター

thisOwner このルールを所有するオブジェクト。ルールは、そのタイプ

によって、キュー・マネージャー、キュー、または他の MQSeries Everyplace オブジェクトを参照することもできます。

戻り値 なし

例外 なし

例

MQeRule close

構文

```
public void close( )
```

説明 このメソッドは、ルールを使用停止にする必要があることを示します。サブクラスにより、このメソッドはオーバーライドされます。

パラメーター

なし

戻り値 なし

例外

例

MQeRule newRule

構文

```
public MQeRule newRule( Object owner, MQeRule thisRule ) throws Exception
```

説明 現行ルールは、所有オブジェクトの振る舞いを指示するために使用されています。このメソッドは、所有オブジェクトで現行ルールを別のルールに変更したい場合に、現行ルールが別のルールにより置き換え可能かどうかを調べるために呼び出します。

デフォルトでは、MQeRule クラスは、ヌル以外の任意の MQeRule オブジェクト参照による置き換えが可能です。

ルールを別のルールで置き換えられないようにする場合は、サブクラスのこのルールをオーバーライドして、たとえば、`MQeException, code=Except_Rule` などの例外を出します。

パラメーター

owner このルールを所有するオブジェクト。現行ルール・クラスにより、このメソッドの振る舞いが決まります。

thisRule 現行ルールを置き換えるかどうかという MQeRule の下位クラスの参照。

戻り値 所有者オブジェクトのアクティブ・ルールとなるルール。

例外

MQeException	Except_Rule このルールが、所有オブジェクトを変更するようにアクティブ・ルールを許可していない場合。

例

```
public MQeRule newRule( Object owner, MQeRule thisRule ) throws Exception   {
// throw new MQeException( MQE.Except_Rule, "Disallowed by rule" );
    if ( thisRule != null )                   /* is there a rule ?            */
          thisRule.activate( owner );          /* activate with new owner      */
     return( thisRule );                       /* use new rule                 */
}
```

MQeEventLogInterface

すべての MQSeries Everyplace ログ・ハンドラーは、このインターフェースを実装する必要があります。

パッケージ **com.ibm.mqe**

メソッド

メソッド	目的
activate	イベント・ログ・ハンドラーをアクティブにします。
close	イベント・ログ機能を終了して、適切な終結処置を実行します。
logOutput	データをイベント・ログへ出力します。

MQeEventLogInterface activate

構文

```
public void activate( String logName ) throws Exception
```

説明 イベント・ログ・ハンドラーをアクティブにするときに呼び出されます。

パラメーター

logName このイベント・ログを示すときに使うストリング。

戻り値 なし

MQeEventLogInterface close

構文

```
public void close( )
```

説明 イベント・ログ・ハンドラーをクローズして、必要な終結処置を実行するときに呼び出されます。

パラメーター
なし

戻り値 なし

MQeEventLogInterface logOutput

構文

```
public void logOutput( String data )
```

説明 メッセージをイベント・ログ・ハンドラーへ出力するときに MQSeries Everyplace によって呼び出されます。

パラメーター

data ログ記録するデータ。

戻り値 なし

MQeMessageListenerInterface

MQeMessage イベントを受け取るオブジェクトはすべて、このインターフェースを実装する必要があります。

パッケージ **com.ibm.mqe**

メソッド

メソッド	目的
messageArrived	MQeMessageEvent.MessageArrived イベントのイベント・ハンドラー。このイベントは、メッセージがキューに到着すると生成されます。

MQeMessageListener messageArrived

構文

```
public void messageArrived( MQeMessageEvent msgEvent )
```

説明 このメソッドは、MQeMessageEvent.MessageArrived イベントが生成されると、すべての listen オブジェクトで呼び出されます。

パラメーター

msgEvent 新しく到着したメッセージの詳細を含む MQeMessageEvent オブジェクト。

戻り値 なし

例外 なし。

例

```
class MyMQeApplication extends MQSeries Everyplace
                       implements MQeMessageListenerInterface
{
  ...
  public void messageArrived( MQeMessageEvent e )
   {
     ...
     if ( e.getQueueName().equals("MY.QUEUE") )
       MQeFields msgFields = e.getMsgFields(); /* get msg info */
     ...
   }
  ...
}
```

MQeRunListInterface

MQSeries Everyplace キュー・マネージャーがアクティブになるときに、それに渡されるパラメーター・セットの一部として、 MQSeries Everyplace アプリケーションの 2 つのリストを MQSeries Everyplace キュー・マネージャーへ渡すことができます。最初のリストに含まれるアプリケーションは、キュー・マネージャーがアクティブになると起動します。 2 番目のリストに含まれるアプリケーションは、キュー・マネージャーがクローズ要求を受け取ると起動します。

すべてのアプリケーションが MQeRunListInterface を実装するようにしますが、必須ではありません。

パッケージ **com.ibm.mqe**

メソッド

メソッド	目的
activate	キュー・マネージャーによって呼び出され、アプリケーションをアクティブにします。

MQeRunListInterface activate

構文

```
public Object activate(  Object    owner,
                         Hashtable loadTable,
                         MQeFields setupData ) throws Exception;
```

説明

MQSeries Everyplace キュー・マネージャーがアクティブになるときに、渡されるパラメーターの一部として、 MQSeries Everyplace アプリケーションの 2 つのリストを MQSeries Everyplace キュー・マネージャーへ渡すことができます。最初のリストには、キュー・マネージャーがアクティブになると起動するアプリケーションが含まれます。 2 番目のリストには、キュー・マネージャーがクローズ要求を受け取ると (**MQeQueueManager.close()** メソッドが呼び出されると) 起動するアプリケーションが含まれます。

キュー・マネージャー・パラメーターに含まれるアプリケーションが MQeRunListInterface を実装している場合、キュー・マネージャーはこのメソッドを呼び出し、アプリケーションをアクティブにして、キュー・マネージャー・パラメーターに含まれるすべてのアプリケーションの設定情報を渡します。

アプリケーションに MQeRunListInterface を実装することが強制されているわけではありませんが、実装されていないと、アプリケーションが起動するだけで、設定情報は何も渡されません。

キュー・マネージャーがアクティブになるときに起動するアプリケーションは、キュー・マネージャーを継続できるように、できるだけすぐにこのメソッドから戻る必要があります。アプリケーションは、呼び出されたスレッドとは別のスレッドで、それ自体を初期設定しなければなりません。このスレッドを管理するのは、アプリケーションの役割です。

キュー・マネージャーのクローズ時にアプリケーションを起動すると、キュー・マネージャーがこのメソッドから戻らずにシャットダウンしてしまうことを防ぐことができます。

パラメーター

owner これは、アプリケーションを所有するオブジェクトです。通常、これは、アプリケーションを起動した MQSeries Everyplace キュー・マネージャーになります。

loadTable キュー・マネージャーによって起動されたアプリケーション間でデータを共用するときに使える java.util.Hashtable オブジェクト。キュー・マネージャーによって起動されるアプリケーションはすべて、このテーブルを参照します。

setupData アプリケーション設定データを含む MQeFields オブジェクト。このデータは、キュー・マネージャーがアクティブにされたときに、キュー・マネージャーへ渡されたパラメーターに含まれていなければなりません。この例については、下記のサンプル INI ファイルを参照してください。

戻り値 オブジェクト参照 - 現在は使われていません。

例外 なし

例 キュー・マネージャーがアクティブにされるときに立ち上げられるアプリケーションの例

```
public class ExampleApp extends MQe implements MQeRunListInterface,
                                                Runnable,
                                                MQeMessageListenerInterface
{
  Thread th = null;
  MQeQueueManager qmgr = null;
  ...
  /* Called by the Queue Manager to activate the application  */
  public Object activate( Object owner, Hashtable loadTable,
                          MQeFields setupData )
  {
    qmgr = (MQeQueueManager)owner; /* QMgr is owner of the application */
    processSetupData( setupData ); /* Process the setup information */
    th = new Thread( this ); /* Create a new thread to listen */
    th.start(); /* for incoming messages */
    return (null); /* return control to the QMgr */
  }

  public void run()
  {
    try
```

```
    {  /* Create a message listener for incoming messages */
       qmgr.addMessageListener( this, "MyQueue", null );
       /* Loop indefintely keeping application alive */
       while( true );
    }
    catch ( Exception e )
    {
      e.printStackTrace( System.err );
    }
  }
  ...
}
```

キュー・マネージャーがクローズ要求を受け取るときに立ち上げられるアプリケーションの例

```
public class ExampleCloseApp extends MQe implements MQeRunListInterface
{
  MQeQueueManager qmgr = null;
  ...
  /* Called by the Queue Manager to activate the application  */
  public Object activate( Object owner, Hashtable loadTable,
                          MQeFields setupData )
  {
    qmgr = (MQeQueueManager)owner; /* QMgr is owner of the application */
    performAction(); /* Perform some action */
    /* don't return control to the QMgr until application has finished */
    return (null);
  }
}
```

キュー・マネージャーのサンプル INI ファイル

```
* Sample Queue Manager INI file

* Queue Manager setup info
[QueueManager]
* Name for this Queue Manager
(ascii)Name=ServerQMgr8082

* Registry setup info
[Registry]
* QueueManager Registry type
(ascii)LocalRegType=com.ibm.mqe.registry.MQePrivateSession
* Location of the registry
(ascii)DirName=d:¥development¥Rename¥Classes¥ServerQMgr8082¥Registry
* Registry access PIN
(ascii)PIN=12345678

* List of applications to launched at Queue Manager activation-time
[ActivateAppList]
(ascii)App1=examples.queuemanager.TestMQeApp
(ascii)App2=examples.administration.AdminApp

* Setup info for App1 - the data in this section is passed to the application
[App1]
(ascii)DefaultMsgPriority = 7
(long)Timeout = 30000

* Setup info for App2 - the data in this section is passed to the application
[App1]
(ascii)DefaultQueueName=AdminReplyQueue
```

MQeSecurityInterface

これは、オプションのインターフェースで、 Java のセキュリティー管理機能によって実装することができます。このインターフェース・メソッドを使うと、セキュリティー管理機能で呼び出しを許可したり拒否できるようになります。 MQSeries Everyplace トレース・ハンドラーは、このインターフェースを実装している必要があります。

パッケージ **com.ibm.MQe**

メソッドの要約

メソッド	サポート
alias	クラス別名が追加されたり削除されるときに呼び出されます。
channelCommand	チャネルによってチャネル・コマンドが処理されるときに呼び出されます。
newAdapter	アダプター定義が定義されるときに呼び出されます。
mapFileDescriptor	ファイル記述子の割り当てを設定するときに呼び出されます。

MQeSecurityInterface alias

構文

```
public void alias( String from, String to ) throws Exception
```

説明 別名を設定したり削除するときに呼び出されます。

パラメーター

from クラス別名を定義するストリング。

to この別名のクラス名を定義するストリング、または、別名が削除される場合はヌル。

戻り値 なし

例外

SecurityException 要求は拒否されました。

MQeSecurityInterface channelCommand

構文

```
public void channelCommand( String command ) throws Exception
```

説明 チャネル・コマンドが処理されるときに呼び出されます。

パラメーター

command チャネル・コマンドを示すストリング。

戻り値 なし

例外

SecurityException	要求は拒否されました。

MQeSecurityInterface newAdapter

構文

```
public void newAdapter( String destination ) throws Exception
```

説明 新しいアダプター定義が設定されるときに呼び出されます。

パラメーター

destination	このアダプターの宛先を示すストリング。通常の宛先は、次のような形式になります。 `Tcpip:127.0.0.1:8080`

戻り値 なし

例外

SecurityException	要求は拒否されました。

MQeSecurityInterface mapFileDescriptor

構文

```
public void mMapFileDescriptor( String fileDesc, Object newDesc )
                                                    throws Exception
```

説明 ファイル記述子の割り当てが設定されるときに呼び出されます。

パラメーター

fileDesc	割り当てられるファイル記述子を示すストリング。
newDesc	割り当てられた記述子であるファイル記述子を示すストリング。

戻り値 なし

例外

SecurityException	要求は拒否されました。

MQeTraceInterface

すべての MQSeries Everyplace トレース・ハンドラーは、このインターフェースを実装する必要があります。

パッケージ **com.ibm.mqe**

メソッド

メソッド	目的
activate	トレース・ハンドラーをアクティブにするときに呼び出されます。
addMessage	新しいトレース・メッセージ・テンプレートを追加します。
addMessageBundle	トレース・メッセージ・テンプレート群を追加します。
close	トレース・ハンドラーをクローズするときに呼び出され、適切な終結処置を実行します。
getMessage	所定のメッセージ番号のメッセージ・テンプレートを戻します。
traceMessage	トレース・メッセージを出力ストリームへ書き込むときに呼び出されます。

MQeTraceInterface activate

構文

```
public void activate ( String title, String resource )
```

説明 トレース・ハンドラーをアクティブにするときに呼び出されます。

パラメーター

title このトレース・ハンドラーのタイトルに使われるストリング、または ヌル。

resource このトレース・ハンドラーで使うリソース群を示すストリング。

戻り値 なし

MQeTraceInterface traceMessage

構文

```
public String traceMessage( String prefix,
                            int msgNumber,
                            Object insert )
```

説明 トレース・ハンドラー経由でトレース・メッセージを出力するときに MQSeries Everyplace によって呼び出されます。

パラメーター

prefix 呼び出すオブジェクト名とインスタンス番号。

msgNumber メッセージ・テンプレートを見つけるときに使うトレース・メッセージ番号を示す整数。

insert メッセージ・テンプレートに適用する挿入。

戻り値 展開されたトレース・メッセージ・テキストを示すストリング。

MQeTraceInterface addMessage

構文

1.
```
public void   addMessage  ( int msgNumber, String msgText ) throws Exception
```
2.
```
public void   addMessage  ( String msgText ) throws Exception
```

説明 新しいトレース・メッセージ・テンプレートをトレース・ハンドラーへ追加するときに呼び出されます。テンプレートの形式は、次のようになります。

```
static final Object[][] contents = {
  /*------------------------------------------------------------------------*/
  /* System messages                                                        */
  /*    ' '         message                                                 */
  /*    'i'         Information                                             */
  /*    'w'         Warning                                                 */
  /*    'e'         Error                                                   */
  /*    'd'         Debug                                                   */
  /*                                                                        */
  /* Application messages                                                   */
  /*    ' '         message                                                 */
  /*    'I'         Information                                             */
  /*    'W'         Warning                                                 */
  /*    'E'         Error                                                   */
  /*    'D'         Debug                                                   */
  /*                                                                        */
  /* Modifier                                                               */
  /*    ':'         no modification applied                                 */
  /*    ';'         RESERVED for create/destroy object                      */
  /*    '+'         Log this message via the Log interface                  */
  /*    ''         ignore - Do not display this message                     */
  /*                                                                        */
  /* Mwssage number                                                         */
  /*    "[nnnnn]:" syntax for message number of this message                */
  /*                                                                        */
  /* Example:                                                               */
  /*            "e+[01000]:Error #0 occurred"                               */
  /*            "I:[01010]:All is wonderful"                                */
  /*------------------------------------------------------------------------*/
```

パラメーター

msgNumber メッセージ・テンプレートを識別するときに使うトレース・メッセージ番号を示す整数。

msgText トレース・メッセージ・テンプレートを示すストリング。

戻り値 なし

MQeTraceInterface addMessageBundle

構文

```
public void addMessageBundle( String msgBundle ) throws Exception
```

説明 テンプレート群をトレース・ハンドラーへ追加するときに呼び出されます。一群のテンプレートは、次のような形式になります。

```
static final Object[][] contents = {
  /*------------------------------------------------------------------------*/
  /* System messages                                                        */
  /*    ' '         message                                                 */
  /*    'i'         Information                                             */
  /*    'w'         Warning                                                 */
  /*    'e'         Error                                                   */
  /*    'd'         Debug                                                   */
  /*                                                                        */
  /* Application messages                                                   */
  /*    ' '         message                                                 */
  /*    'I'         Information                                             */
  /*    'W'         Warning                                                 */
  /*    'E'         Error                                                   */
  /*    'D'         Debug                                                   */
  /*                                                                        */
  /* Modifier                                                               */
  /*    ':'         no modification applied                                 */
  /*    ';'         RESERVED for create/destroy object                      */
  /*    '+'         Log this message via the Log interface                  */
  /*    ''         ignore - Do not display this message                     */
  /*                                                                        */
  /* Mwssage number                                                         */
  /*    "[nnnnn;]:" syntax for message number of this message               */
  /*                                                                        */
  /* Example:                                                               */
  /*            "e+[01000]:Error #0 occurred"                               */
  /*            "I:[01010]:All is wonderful"                                */
  /*------------------------------------------------------------------------*/
```

パラメーター

msgBundle
追加するトレース・メッセージ・テンプレート群を示すストリング。

戻り値 なし

標準的なトレース・メッセージ

```
static final Object[][] contents = {
  /*------------------------------------------------------------------------*/
  /* System messages                                                        */
  /*    ' '         message                                                 */
  /*    'i'         Information                                             */
  /*    'w'         Warning                                                 */
  /*    'e'         Error                                                   */
  /*    's'         Security                                                */
  /*    'd'         Debug                                                   */
  /*                                                                        */
  /* Application messages                                                   */
  /*    ' '         message                                                 */
  /*    'I'         Information                                             */
```

```
/*    'W'        Warning                                                 */
/*    'E'        Error                                                   */
/*    'S'        Security                                                */
/*    'D'        Debug                                                   */
/*                                                                       */
/* Modifier                                                             */
/*    ':'        no modification applied                                 */
/*    ';'        RESERVED for create/destroy object                      */
/*    '+'        Log this message via the Log interface                  */
/*    ''        ignore - Do not display this message                     */
/*                                                                       */
/* Mwssage number                                                       */
/*    "[nnnnn]:" syntax for message number of this message              */
/*                                                                       */
/* Example:                                                             */
/*            "e+[01000]:Error #0 occurred"                             */
/*            "I:[01010]:All is wonderful"                              */
/*----------------------------------------------------------------------*/

/* common messages                                                      */
       {    "1",     "d;[00001]:Created"    },
       {    "2",     "d;[00002]:Destroyed"  },
       {    "3",     "d:[00003]:Close"      },
       {    "4",     "w:[00004]:Warning:#"  },
       {    "5",     "e:[00005]:Error:#"    },
       {    "6",     "i:[00006]:Command:#"  },
       {    "7",     "i:[00007]:Waiting"    },
       {    "8",     "i:[00008]:# input byte count=#"  },
       {    "9",     "i:[00009]:# output byte count=#" },
/* com.ibm.MQe.MQeLoader                                                */
       {   "10",     "d:[00010]:loadClass #"           },
       {   "11",     "d:[00011]:Loaded (bytes) #"      },
       {   "12",     "d:[00012]:Resolved Class #"      },
       {   "13",     "d:[00013]:DropClass #0"          },
/* com.ibm.MQe.MQeChannel & ChannelManager                              */
       {   "20",     "d:[00020]:ActivateMaster"              },
       {   "21",     "d:[00021]:ActivateSlave"               },
       {   "22",     "d:[00022]:ActivateSlave Channel ID=#0" },
       {   "23",     "d:[00023]:Close Channel ID=#0"         },
       {   "24",     "d:[00024]:SlaveResponse"               },
       {   "25",     "d:[00025]:SlaveResponse Channel ID=#0" },
       {   "26",     "i:[00026]:Timeout channel ID=#0"       },
       {   "27",     "i:[00027]:Forwarding to #0"            },
       {   "28",     "i:[00028]:ID=#0, Command=#1"           },
/* com.ibm.MQe.MQeChannelListener                                       */
       {   "30",     "i:[00030]:Starting Listener #0"        },
       {   "31",     "i:[00031]:Stopping Listener"           },
       {   "32",     "d:[00032]:Starting Slave"              },
       {   "33",     "d:[00033]:Stopping Slave"              },
       {   "34",     "d:[00034]:Timer interval"              },
/* com.ibm.MQe.MQeAttribute                                             */
       {   "40",     "d:[00040]:Authenticator #0"            },
       {   "41",     "d:[00041]:Compressor #0"               },
       {   "42",     "d:[00042]:Cryptor #0"                  },
       {   "43",     "d:[00043]:Attribute(Rule).equals=#0"   },
```

```
        {   "44",     "d:[00044]:Attribute Change #0"         },
        {   "45",     "d:[00045]:TargetRegistry=#0"           },
        {   "46",     "s:[00046]:Secure Chnl State=pending, KeyObject=#0"  },
  /* com.ibm.MQe.MQeTransporter                                            */
        {   "50",     "d:[00050]:#0 PID=#1"                   },
        {   "51",     "d:[00051]:#0 made persistent PID=#1"   },
        {   "52",     "d:[00052]:#0 Message Request for Queue '#1'"  },
        {   "53",     "d:[00053]:GetPendingMessage for Queue Manager '#0'" },
  /* ****** Adapters ******                                                 */
  /* com.ibm.MQe.Adapters.MQeIniFileAdapter & MQeDisk....Adapter           */
        {  "110",     "d:[00110]:Object #0 - saved"               },
        {  "111",     "d:[00111]:Object #0 - loaded"              },
        {  "112",     "d:[00112]:Object #0 - Selected"            },
        {  "113",     "d:[00113]:Object #0 - matched"             },
        {  "114",     "d:[00114]:Object #0 - deleted"             },
  /* com.ibm.MQe.Adapters.MQeTcpipAdapter                                  */
        {  "200",     "d:[00200]:File descriptor '#0'"   },
        {  "201",     "d:[00201]:Socket pending"         },
        {  "202",     "d:[00202]:Control '#0'"           },
  /* com.ibm.MQe.Adapters.MQeTcpipHttpAdapter                              */
        {  "203",     "d:[00203]:Read Header"            },
        {  "204",     "d:[00204]:Header: #0"             },
        {  "205",     "d:[00205]:Header length=#0"       },
        {  "206",     "d:[00206]:Write Header"           },
        {  "207",     "d:[00207]:Read Content-length=#0" },
        {  "208",     "d:[00208]:Readln #0"              },
  ...
  ...
              };
```

メッセージの完全なリストは、 examples ディレクトリーの trace サブディレクトリーに含まれる examples.trace.MQeTraceResource.java ソース・ファイルにあります。

MQeTraceInterface close

構文

説明

パラメーター

戻り値

MQeTraceInterface getMessage

構文

```
public String getMessage( int msgNumber )
```

説明 *msgNumber* パラメーターに指定したトレース・メッセージ番号に対応するストリングを取得するときに呼び出されます。

パラメーター

msgNumber 戻されるトレース・メッセージ・ストリングの番号。

戻り値　トレース・メッセージ・テンプレートを示すストリング。

第3章 com.ibm.mqe.administration のクラス

この節には、以下の MQSeries Everyplace クラスおよびインターフェースについての詳細が載せられています。

表 13. パッケージ com.ibm.mqe.Administration のクラス

クラスまたはインターフェース名	目的
MQeAdminQueueAdminMsg	MQeAdminQueue タイプのキューを管理するために使用されます
MQeConnectionAdminMsg	MQeConnectionDefinition タイプの接続を管理するために使用されます
MQeHomeServerQueueAdminMsg	MQeHomeServerQueue タイプのキューを管理するために使用されます
MQeQueueAdminMsg	MQeQueue タイプの MQSeries Everyplace ローカル・キューを管理するために使用されます
MQeQueueManagerAdminMsg	MQeQueueManager タイプのキュー・マネージャーを管理するために使用されます
MQeRemoteQueueAdminMsg	MQeRemoteQueue タイプのリモート・キューを管理するために使用します
MQeStoreAndForwardQueueAdminMsg	MQeStoreAndForwardQueue タイプのキューを管理するために使用します

MQeAdminQueueAdminMsg

このクラスは、MQeAdminQueue タイプのキューを管理するために使用します。このクラスにより MQeQueueAdminMsg が拡張され、管理キューを管理するための実装が可能になります。

このキューを使用することにより、MQSeries Everyplace 管理対象リソースの管理を、ローカルであるかリモートであるかを意識せずに行うことができます。

パッケージ　　com.ibm.mqe.administration

このクラスは、MQeQueueAdminMsg の下位クラスです。

定数と変数

MQeAdminQueueAdminMsg には、 MQeQueueAdminMsg によって提供され継承される定数と変数だけでなく、以下の定数と変数が備えられています。

キューの特性

QtimerInterval;

(ミリ秒 (long) の) 間隔を置いて未解決の管理メッセージを処理します。

```
public final static String Queue_QtimerInterval;
```

MQeConnectionAdminMsg

このクラスは、MQeConnectionDefinition タイプの接続を管理するために使用します。

このクラスにより MQeAdminMsg が拡張され、接続を管理するための実装が可能になります。接続については、以下のアクションを適用できます。

- **Action_Create**
- **Action_Delete**
- **Action_Inquire**
- **Action_InquireAll**
- **Action_Update**

接続では、特定のキュー・マネージャーが別のキュー・マネージャーへの接続を確立する方法を定義します。接続に関連付けられた主な特性は、以下のとおりです。

使うチャネルのタイプ
以下のタイプのチャネルが用意されています。

MQeChannel
クライアントからサーバーまたはサーバーからサーバーのチャネル。

MQePeerChannel
クライアントからクライアントのチャネル。

通信アダプターとそのパラメーターを含むファイル記述子
以下のアダプターが用意されています。

MQeTcpipLengthAdapter
簡単な TCPIP アダプター。

MQeTcpipHistoryAdapter
永続的な接続およびデータ圧縮を可能にする TCPIP アダプター。

MQeTcpipHttpAdapter
HTTP アダプター。

ポート 8082 でサーバー 192.168.0.1 へ HTTP 接続する場合、ファイル記述子は次のように定義されます。

```
com.ibm.mqe.Adpaters.MQeTcpipHttpAdapter:192.168.0.1:8082
```

HTTP アダプター用に ″Network″ の別名が設定されている場合は、次のファイル記述子を使うことができます。

```
Network:192.168.0.1:8081
```

パッケージ **com.ibm.mqe.administration**

MQeConnectionAdminMsg

このクラスは、MQeAdminMsg の下位クラスです。

定数と変数

MQeConnectionAdminMsg には、MQeAdminMsg により提供される定数と変数だけでなく、以下の定数と変数が用意されています。

その他のアクション

AddAlias

接続の別名を追加します。

```
public final static  int    Action_AddAlias
```

RemoveAlias

接続の別名を削除します。

```
public final static  int    Action_RemoveAlias
```

接続の特性

アダプター (フィールド配列)

ターゲット・キュー・マネージャーに接続するときに使用するアダプターのセット。

注: 現行リリースでは、1 つのアダプターだけが許可されています。

```
public final static  String     Con_Adapters
```

以下のフィールドでは、*Adapters* 配列の各要素で許可されるアダプターを定義します。

AdapterFileDesc (ASCII)

```
public final static  String     Con_Adapter
```

アダプター・ファイル記述子。たとえば、Network:192.168.0.3:8085。

AdapterAsciiParm (ASCII)

```
public final static  String     Con_AdapterAsciiParm
```

アダプターのパラメーター。

AdapterEncodedParm (バイト配列)

```
public final static  String     Con_AdapterEncodedParm
```

エンコードされたパラメーター。

AdapterOptions (ASCII)

```
public final static String Con_AdapterOptions
```

| アダプターのオプション。たとえば、<HISTORY>。

| 上記の各フィールドは、使用するアダプターによって異なります。標準
| MQSeries Everyplace アダプターがそれぞれ使用するフィールドに関して、詳し
| くは、439ページの『第9章 com.ibm.mqe.adapters のクラス』を参照してくださ
| い。これらのフィールドは、その **activate()** メソッドを介して各アダプター・
| オブジェクトに渡されます。

別名 この接続の別名群 (ASCII 配列)。

```
public final static String    Con_Aliases
```

チャネル

チャネル・クラス (ASCII) - この接続で使用するチャネルのタイプ。次に例を示します。

```
com.ibm.mqe.MQeChannel
com.ibm.mqe.MQePeerChannel
```

値はヌルになる場合もあり、これはローカル接続であることを意味します。

```
 public final static  String   Con_Channel
```

説明 接続の記述 (Unicode)。

```
public final static  String   Con_Description
```

メソッド

メソッド	目的
create	Action_Create アクションを実行する管理メッセージを設定します。
update	Action_Update アクションを実行する管理メッセージを設定します。

MQeConnectionAdminMsg create

構文

```
public void create( String adapter, String parameters,
                    String options, String channel
                    String description ) throws Exception
```

説明 **Action_Create** アクションを実行する管理メッセージを設定します。このバージョンの **create()** は、1 つのアダプターを持つ接続のために、簡単な接続定義を追加します。

パラメーター

adapter アダプターのファイル記述子。

parameters アダプターのパラメーター。

options アダプターのオプション。

channel 使うチャネルのタイプ。

description 接続の記述。

戻り値 なし

例外

java.lang.Exception さまざまなものがあります。

例

```
class MyApplication
{
  ...
  MQeConnectionAdminMsg con = new MQeConnectionAdminMsg()
  con.setName("ServerQM123");
  con.create("Network:127.0.0.1:8081",
                      null,
                      null.
                      "DefaultChannel",
                      "Con to MQeServer" );
  MQeConnectionAdminMsg con2 = new MQeConnectionAdminMsg()
  con2.setName("ServletQM123");
  con2.create("Network:127.0.0.1:8081",
                      "/servlet/MQe",
                      null,
                      "DefaultChannel",
                      "Con to MQeServlet" );
  ...
}
```

MQeConnectionAdminMsg update

構文

```
public void update( String adapter, String parameters,
                    String options, String channel
                    String description ) throws Exception
```

説明 **Action_Update** アクションを実行する管理メッセージを設定します。このバージョンの **update()** は、1 つのアダプターを持つ接続のために、既存の接続定義を簡単な接続定義に置き換えます。

パラメーター

adapter アダプターのファイル記述子。

parameters アダプターのパラメーター。

options アダプターのオプション。

channel 使うチャネルのタイプ。

description 接続の記述。

戻り値 なし

例外

java.lang.Exception	さまざまなものがあります。

例

```
class MyApplication
{
  ...
  MQeConnectionAdminMsg con = new MQeConnectionAdminMsg()
  con.setName("ServerQM123");
  con.update("Network:127.0.0.1:8082",
                      null,
                      null,
                      "DefaultChannel",
                      "Con to MQeServer" );
...
}
```

MQeHomeServerQueueAdminMsg

このクラスは、MQeHomeServerQueue タイプのキューを管理するときに使用します。このクラスにより MQeRemoteQueueAdminMsg が拡張され、ホーム・サーバー・キューを管理するための実装が可能になります。

このキューは、そのホーム・サーバーのホーム・キューから (バックグラウンド・スレッドを使用して) 保留メッセージを取り出すために、クライアントで使用されます。

パッケージ **com.ibm.mqe.administration**

このクラスは、 MQeRemoteQueueAdminMsg の下位クラスです。

定数と変数

MQeHomeServerQueueAdminMsg には、 MQeRemoteQueueAdminMsg によって提供され継承される定数と変数だけでなく、以下の定数と変数が備えられています。

キューの特性

QtimerInterval;

(ミリ秒 (long) の) 間隔を置いて保留メッセージを処理します。

```
public final static String Queue_QtimerInterval;
```

MQeQueueAdminMsg

このクラスは、MQeQueue タイプの MQSeries Everyplace ローカル・キューを管理するために使用します。このクラスにより MQeAdminMsg が拡張され、ローカル・キューを管理するための実装が可能になります。ローカル・キューについては、以下のアクションを適用できます。

- Action_Create
- Action_Delete
- Action_Inquire
- Action_InquireAll
- Action_Update
- Action_AddAlias
- Action_RemoveAlias

名前が示すように、ローカル・キューは、所有するキュー・マネージャーに対してローカルです。キューの格納場所と格納位置を示すファイル記述子を設定する必要があります。これは、アダプターと、そのアダプターに対するパラメーターの 2 つの部分で構成されます。以下のアダプターが用意されています。

MQeDiskFieldsAdapter
MQeFields オブジェクトのためのファイル・ベースのアダプター。

MQeMemoryFieldsAdapter
MQeFields オブジェクトのためのメモリー・ベースのアダプター。

たとえば、MQeDiskFieldsAdapter に別名 MsgLog が設定され、メッセージを d:¥ServerQM123¥Queues に格納する場合、ファイル記述子は次のようになります。

```
MsgLog:d:¥ServerQM123¥Queues
```

キューでは、基本となるローカル・キューによって使われない、複数の特性を設定することができます。このような特性は、ユーザーが置き換え可能なキュー・ルール・クラスで使えるようになるので、最大限に利用することができます。たとえば、*Queue_MaxQSize* を設定できますが、MQeQueue によって検査されません。最大キュー・サイズの妥当性検査を実行することは、キュー・ルール・クラスの役割です。

このクラスは、他のタイプのキューを管理するときの基本クラスとして機能します。たとえば、MQeRemoteQueueAdminMsg はこのクラスから派生するものであり、リモート・キューの管理を扱います。

パッケージ　**com.ibm.mqe.Administration**

このクラスは、MQeAdminMsg の下位クラスです。

定数と変数

MQeQueueAdminMsg には、 MQeAdminMsg により提供される定数と変数だけでなく、以下の定数と変数が用意されています。

その他のアクション

AddAlias

キューの別名を追加します。

```
public final static  int    Action_AddAlias;
```

RemoveAlias

キューの別名を削除します。

```
public final static  int    Action_RemoveAlias;
```

キューの特性

Active キューがアクティブであることを示します (ブール値、読み取り専用)。

```
public final static  String   Queue_Active
```

CreationDate

日付キューが作成されました (long 型で読み取り専用)。 1970 年 1 月 1 日午前零時 (GMT) 以降に経過した時間をミリ秒で表します。

```
public final static  String   Queue_CreationDate
```

CurrentSize

現行キュー項目数 (int 型で読み取り専用)。

```
public final static  String   Queue_CurrentSize
```

Description

キューの記述 (Unicode)。

```
public final static  String   Queue_Description
```

Expiry キューのメッセージは、キューに格納されてから n ミリ秒が経過しました (long)。

```
public final static  String   Queue_Expiry
```

FileDesc

ファイル記述子 (キューが格納されている場所) (ASCII)。

ファイル記述子をいったん設定したら変更できません。

ファイル記述子は、以下の 2 つの部分で構成されます。

- アダプター
- アダプターのパラメーター

たとえば、MQeDiskFieldsAdapter に別名 MsgLog が設定され、メッセージを d:¥ServerQM123¥Queues に格納する場合、ファイル記述子は MsgLog:d:¥ServerQM123¥Queues になります。

```
public final static  String   Queue_FileDesc
```

MaxMsgSize

キューに許可されているメッセージの最大長 (int)。

```
public final static  String   Queue_MaxMsgSize
```

MaxQSize

キューに許可されているメッセージの最大数 (int)。

NoLimit

```
public final static  String   Queue_MaxQSize
public final static  int    Queue_NoLimit
```

Mode キューのタイプ - キュー同期またはキュー非同期です。

```
public final static  String   Queue_Mode
```

Asynchronous

キューは非同期です。

```
public final static  byte    Queue_Asynchronous
```

Synchronous

キューは同期です。

```
public final static  byte    Queue_Synchronous
```

Priority

まだメッセージに指定していなければ、そのメッセージのデフォルトの優先順位 (バイト) (最小 = 0、最大 = 9)。

```
public final static  String   Queue_Priority
```

QAliasNameList

このキューに別名を設定します (ASCII 配列)。

```
public final static String    Queue_QAliasNameList
```

QMgrName

キューを所有するキュー・マネージャーの名前 (ASCII)。キュー・マネージャー名をいったん設定したら変更できません。

```
public final static  String   Queue_QMgrName
```

キューのセキュリティー特性

これらのフィールドを変更できるのは、キューにメッセージがなく、アクティブでないときにだけです。

AttrRule

キューの属性ルール・クラスの名前 (ASCII)

```
public final static  String   Queue_AttrRule
```

Authenticator
オーセンティケーター・クラスの名前 (ASCII)

```
public final static  String   Queue_Authenticator
```

Compressor
圧縮機能クラスの名前 (ASCII)

```
public final static  String    Queue_Compressor
```

Cryptor
暗号機能クラスの名前 (ASCII)

```
public final static  String    Queue_Cryptor
```

TargetRegistry
ターゲット・レジストリーのタイプ (バイト)。

```
public final static  String   Queue_TargetRegistry
```

以下のいずれかになります。

RegistryNone

```
public final static  byte    Queue_RegistryNone
```

RegistryQMgr

```
public final static  byte    Queue_RegistryQMgr
```

RegistryQueue

```
public final static  byte    Queue_RegistryQueue
```

Rule キュー・ルール・クラスの名前 (ASCII)

```
public final static  String   Queue_Rule
```

コンストラクター

MQeQueueAdminMsg

構文

1. `public MQeQueueAdminMsg() throws Exception`
2. `public MQeQueueAdminMsg( String qMgrName, String queueName )`

説明 2 つのコンストラクターがあります。

1. このバージョンは、デフォルトの MQeAdminMsg を作成して初期化します。
2. このバージョンは、管理されるキューの名前を使います。

パラメーター

qMgrName キュー・マネージャー名

queueName キュー名

戻り値 新しい MQeQueueAdminMsg。

例外

java.lang.Exception さまざまなものがあります。

例

```
class MyApplication
{
  MQeQueueAdminMsg aMsg = new MQeQueueAdminMsg( "ExampleQM", "ExampleQ" );
}
```

メソッド

コンストラクター	目的
addAlias	Admin_AddAlias アクションを実行する管理メッセージを設定します。
removeAlias	Admin_RemoveAlias アクションを実行する管理メッセージを設定します。
setName	アクションを実行するときの対象となるキューの名前を設定します。

MQeQueueAdminMsg addAlias

構文

```
public void addAlias( String aliasName ) throws Exception
```

説明 **Admin_AddAlias** アクションを実行する管理メッセージを設定します。 1 つのキューに対して、別名を何も持たないか、1 つの別名、または複数の別名を持つことができます。このメソッドは、 1 つの管理メッセージに複数の別名を追加できるように、何度も呼び出すことができます。

パラメーター

aliasName キューの別名。

戻り値 なし

例外

java.lang.Exception さまざまなものがあります。

例

```
class MyApplication
{
  ...
  // Add aliases to a queue
  MQeQueueAdminMsg msg = new MQeQueueAdminMsg();
  msg.setName( "ExampleQM", "ExampleQ" );
```

```
  // Set the action required and its parameters
  // into the message
  msg.addAlias( "PayrollQ" );
  msg.addAlias( "Branch1PayrollQ" );
  ...
}
```

MQeQueueAdminMsg removeAlias

構文

```
public void removeAlias( String aliasName ) throws Exception
```

説明 Admin_RemoveAlias アクションを実行する管理メッセージを設定します。このアクションは、指定した別名をキューから削除します。このメソッドは、1 つの管理メッセージを使って複数の別名を削除できるように、何度も呼び出すことができます。

パラメーター

aliasName キューの別名。

戻り値 なし

例外

java.lang.Exception さまざまなものがあります。

例

```
class MyApplication
{
  ...
  // Add aliases to a queue
  MQeQueueAdminMsg msg = new MQeQueueAdminMsg();
  msg.setName( "ExampleQM", "ExampleQ" );

  // Set the action required and its parameters
  // into the message
  msg.removeAlias( "PayrollQ" );
  msg.removeAlias( "Branch1PayrollQ" );
  ...
}
```

MQeQueueAdminMsg setName

構文

```
public void setName( String qMgrName, String queueName ) throws Exception
```

説明 アクションを実行するときの対象となるキューの名前を設定します。

パラメーター

qMgrName キューを所有するキュー・マネージャーの名前。

queueName キューの名前。

戻り値 なし

例外

java.lang.Exception	さまざまなものがあります。

例

```
class MyApplication
{
  ...
  // Delete a queue
  MqeFields parms = new MQeFields();

  // Set the action required and its parameters
  // into the message
  msg.delete( parms );
  msg.setName( "ExampleQM", "ExampleQ" );
  ....
}
```

MQeQueueManagerAdminMsg

この節では、基本の MQeQueueManagerAdminMsg を作成するために使う Java クラスについて説明します。

このクラスにより MQeAdminMsg が拡張され、 MQSeries Everyplace キュー・マネージャーを管理するための実装が可能になります。キュー・マネージャーについては、以下のアクションを適用できます。

- Action_Inquire
- Action_InquireAll
- Action_Update

注: キュー・マネージャーを正しい位置に置き、管理が実行される前にキュー・マネージャー上で管理キューを初期設定しておく必要があるため、キュー・マネージャーでは、**作成**および**削除**アクションはサポートされていません。また、 Java 仮想マシンごとに 1 つのキュー・マネージャーだけがサポートされます。

パッケージ　　com.ibm.mqe.administration

このクラスは、MQeAdminMsg の下位クラスです。

定数と変数

MQeQueueManagerAdminMsg には、 MQeAdminMsg によって提供され継承される定数と変数だけでなく、以下の定数と変数が備えられています。

キュー・マネージャーの特性

ChnlAttrRules

チャネル属性ルール。

```
public final static  String   QMgr_ChnlAttrRules
```

ChnlTimeout

チャネルをオープンしておくミリ秒単位での最大時間数 (long)。

```
public final static  String    QMgr_ChnlTimeout
```

Connections

キュー・マネージャーが認識している接続 (ASCII 配列 - 読み取り専用)。

```
public final static  String   QMgr_Connections
```

Description

キュー・マネージャーの記述 (Unicode)。

```
public final static  String   QMgr_Description
```

Queues

キュー・マネージャーが認識しているキュー (フィールド配列 - 読み取り専用)。

```
public final static  String     QMgr_Queues
```

QueueName

キュー名 (ASCII)。

```
public final static  String     QMgr_QueueName
```

QueueQMgrName

キュー・マネージャー名 (ASCII)。

```
public final static  String     QMgr_QueueQMgrName
```

QueueType

キューのタイプ (ASCII)。

```
public final static  String     QMgr_QueueType
```

Rules キュー・マネージャーの機能を制御するユーザー置き換え可能なルール (ASCII)。

```
public final static  String   QMgr_Rules
```

MQeRemoteQueueAdminMsg

このクラスは、MQeRemoteQueue タイプのリモート・キューを管理するために使用します。このクラスにより MQeQueueAdminMsg が拡張され、リモート・キューを管理するための実装が可能になります。

リモート・キューには、2 つのタイプがあります。

Synchronous
同期アクセス向けに設定されたリモート・キューに要求が出されると、チャネルは、キューがローカルであるノードに向けてオープンされます。そのため、要求が出されると、同期キューでのアクションはすべて、キューがローカルである場所に移されます。さらに、要求が出されるときには、リモート・キューとローカル・キューとの間でネットワーキング機能が使えなければなりません。

Asynchronous
非同期アクセス向けに設定されたリモート・キューに要求が出されると、メッセージは一時的にそのリモート・キューに格納されます。ユーザー置き換え可能なルールに基づき、将来のある時点で、メッセージはリモート・キューからキューがローカルである場所に移動します。そのため、リモート非同期キューには、メッセージを移動する前に格納しておく場所を記述したファイル記述子が必要になります。

パッケージ **com.ibm.mqe.administration**

このクラスは、MQeQueueAdminMsg の下位クラスです。

定数と変数

MQeRemoteQueueAdminMsg には、 MQeQueueAdminMsg によって提供され継承される定数と変数だけでなく、以下の定数と変数が用意されています。

キューの特性

Transporter
使用する移送機能クラスの名前 (ASCII)。

DefaultTransporter

```
public final static  String   Queue_Transporter
public final static  String   Queue_DefaultTransporter
```

TransporterXOR
移送機能で xor 圧縮を使えるようにします (ブール)。

移送機能で移動した以前のメッセージに含まれる同じ名前のフィールド (あれば) を使って、メッセージの各フィールドを XOR 演算することにより、ネッ

トワーク経由でメッセージを移動するときに、それらのメッセージをインテリジェントに圧縮できるようになります。

```
public final static  String   Queue_TransporterXOR
```

MQeStoreAndForwardQueueAdminMsg

このクラスは、 MQeStoreAndForwardQueue タイプのキューを管理するために使用します。このクラスにより MQeRemoteQueueAdminMsg が拡張され、ストア・アンド・フォワード・キューを管理するための実装が可能になります。

このタイプのキューは、中間ノードにおいて、単に通過するだけのメッセージ、つまり最終宛先がこのシステム上のどのキューでもないメッセージを保持するために使用されます。

パッケージ **com.ibm.mqe.administration**

このクラスは、 MQeRemoteQueueAdminMsg の下位クラスです。

定数と変数

MQeStoreAndForwardQueueAdminMsg には、 MQeRemoteQueueAdminMsg によって提供されたり継承されたりする定数と変数のほかに、以下の定数と変数が用意されています。

その他のアクション

AddQueueManager

キュー・マネージャーを追加します。

```
public final static  int    Action_AddQueueManager;
```

RemoveQueueManager

キュー・マネージャーを削除します。

```
public final static  int    Action_RemoveQueueManager;
```

リモート・キューの特性

QMgrNameList

ストア・アンド・フォワード・キュー・メッセージによって処理されるターゲット・キュー・マネージャー用のキュー・マネージャー・ターゲットのリストは、メッセージがターゲット・キュー・マネージャーによって収集されるまで、または通信が確立されるまで、一時的にこのキューで保管されます。

```
public final static  String     Queue_QMgrNameList
```

メソッド

メソッド	目的
addQueueManager	**Admin_AddQueueManager** アクションを実行するための管理メッセージを設定します。
removeQueueManager	**Admin_RemoveQueueManager** アクションを実行するための管理メッセージを設定します。

MQeStoreAndForwardQueueAdminMsg addQueueManager

構文

```
public void addQueueManager( String targetQMgrName ) throws Exception
```

説明 **Admin_AddQueueManager** アクションを実行するための管理メッセージを設定します。 1 つのキューに対して、 1 つまたは複数のターゲット・キュー・マネージャーを設定できます。このメソッドを複数回呼び出せば、 1 つの管理メッセージで複数のターゲットを追加できます。

パラメーター

targetQMgrName
ターゲット・キュー・マネージャー名

戻り値 なし

例外

java.lang.Exception いろいろな場合

例

```
class MyApplication
{
  ...
  // Add target queue managers to a S&F queue
  MQeStoreAndForwardQueueAdminMsg msg =
                              new MQeStoreAndForwardQueueAdminMsg();
  msg.setName( "ExampleQM", "ExampleQ" );

  // Set the action required and its parameters
  // into the message
  msg.addQueueManager( "Client129345" );
  msg.addQueueManager( "Client129387" );
  ...
}
```

MQeStoreAndForwardQueueAdminMsg removeQueueManager

構文

```
public void removeQueueManager( String targetQMgrName ) throws Exception
```

説明 **Admin_RemoveQueueManager** アクションを実行するための管理メッセージを設定します。このメソッドを複数回呼び出せば、 1 つの管理メッセージで複数のターゲットを削除できます。

パラメーター

targetQMgrName ターゲット・キュー・マネージャー名

戻り値 なし

例外

java.lang.Exception	いろいろな場合

例

```
class MyApplication
{
  ...
  // Remove target queue managers from S&F queue
  MQeStoreAndForwardQueueAdminMsg msg =
                                new MQeStoreAndForwardQueueAdminMsg();
  msg.setName( "ExampleQM", "ExampleQ" );

  // Set the action required and its parameters
  // into the message
  msg.removeQueueManager( "Client129345" );
  msg.removeQueueManager( "Client129387" );
  ...
}
```

第4章 com.ibm.mqe.attributes のクラス

この節には、以下の MQSeries Everyplace クラスについての詳細が載せられています。

表 14. パッケージ com.ibm.mqe.attributes のクラス

クラス名	目的
MQe3DESCryptor	3DES 暗号化のメカニズムを提供します。
MQeDESCryptor	DES 暗号化のメカニズムを提供します。
MQeGenDH	ソリューション固有の MQeDHk クラス・オブジェクトを作成するときの基になる MQeDHk.java ファイルを作成します。
MQeListCertificates	MQeListCertificates オブジェクトを構成します。
MQeLocalSecure	簡単なローカル・セキュリティー・サービスを提供します。
MQeLZWCompressor	LZW 圧縮のメカニズムを提供します。
MQeMARSCryptor	MARS 暗号化のメカニズムを提供します。
MQeMAttribute	簡単なメッセージ・レベルの保護を提供します。
MQeMTrustAttribute	さらに高機能なメッセージ・レベルの保護を提供します。
MQeRC4Cryptor	RC4 暗号化のメカニズムを提供します。
MQeRC6Cryptor	RC6 暗号化のメカニズムを提供します。
MQeRleCompressor	Run Length エンコード圧縮のメカニズムを提供します。
MQeWTLSCertAuthenticator	ミニ認証のメカニズムを提供します。
MQeXORCryptor	XOR 暗号化のメカニズムを提供します。

MQe3DESCryptor

このクラスは、3 つ組の DES 暗号機能オブジェクトを作成するときに使います。このオブジェクトは、特定の属性オブジェクトによって使われるときに、その属性オブジェクトに 3 つ組の DES 暗号機能を実行するメカニズムを提供します。属性オブジェクトは、チャネルおよび MQeFields オブジェクトに関連付けられています。

パッケージ **com.ibm.mqe.attributes**

このクラスは、MQeCryptor の下位クラスです。

コンストラクター

MQe3DESCryptor

構文

```
public MQe3DESCryptor( )
```

説明 MQe3DESCryptor オブジェクトを構成します。

パラメーター

なし

戻り値 なし

例外

MQeException	Except_S_Cipher, ″cip3DES, wrong cipher or key″

例

```
try
    {
    MQe3DESCryptor tripledes   = new MQe3DESCryptor();
    MQeAttribute tripledesA    = new MQeAttribute(null, tripledes, null);
    ...
    }
  catch ( Exception e )
    {
    ...
    }
```

関連する関数

- MQeCryptor
- MQeAttribute
- MQeLocalSecure
- MQeMAttribute
- MQeMTrustAttribute

MQeDESCryptor

このクラスは、DES 暗号機能オブジェクトを作成するときに使います。このオブジェクトは、特定の属性オブジェクトによって使われるときに、その属性オブジェクトに DES 暗号機能を実行するメカニズムを提供します。属性オブジェクトは、チャネルおよび MQeFields オブジェクトに関連付けられています。

パッケージ **com.ibm.mqe.attributes**

このクラスは、MQeCryptor の下位クラスです。

コンストラクター

MQeDESCryptor

構文

```
public MQeDESCryptor( )
```

説明 MQeDESCryptor オブジェクトを構成します。

パラメーター

なし

戻り値 なし

例外

MQeException	Except_S_Cipher, ″cipDES, wrong cipher or key″

例

```
try
  {
  MQeDESCryptor des   = new MQeDESCryptor();
  MQeAttribute desA   = new MQeAttribute(null, des, null);
  ...
  }
catch ( Exception e )
  {
  ...
  }
```

関連する関数

- MQeCryptor
- MQeAttribute
- MQeLocalSecure
- MQeMAttribute
- MQeMTrustAttribute

MQeGenDH

このクラスは、ソリューション固有の MQeDHk クラス・オブジェクトを作成するときの基になる MQeDHk.java ファイルを作成するときに使います。

パッケージ **com.ibm.mqe.attributes**

このクラスは MQe の下位クラスです。

メソッドの要約

メソッド	目的
main	ユーティリティーを呼び出して新しい MQDHk.java ファイルを作成します。
genParams	新しい DH のペアを生成し、その新しいペアを使って新しい MQeDHk.java ファイルを作成します。

セキュア・チャネル設定に流れるデータ部分は、 Diffie Hellman 部分キー・データです。このデータは、チャネル暗号化の暗号化メソッドと復号メソッドで使われ、共有秘密鍵が生成され (続いてその派生物が使われる)、チャネル・データの秘密性が保護されます。 例には、ユーティリティーを使って、 512 ビットの Diffie Hellman 鍵ペアを作成する方法が示されています。このことを MQSeries Everyplace で使えるようにするためには、生成された MQeDHk.java ファイルをコンパイルし、 com.ibm.mqe.attributes パッケージの一部としてインストールする必要があります。

MQeGenDH main

構文

```
java com.ibm.mqe.attributes.MQeGenDH <parameter1><parameter2>
```

説明 ユーティリティーを呼び出して新しい DH 鍵ペアを生成し、新しい MQeDHk.java ファイルを作成します。

パラメーター

<parameter 1> 任意指定。DH パラメーター長で、デフォルトは 512

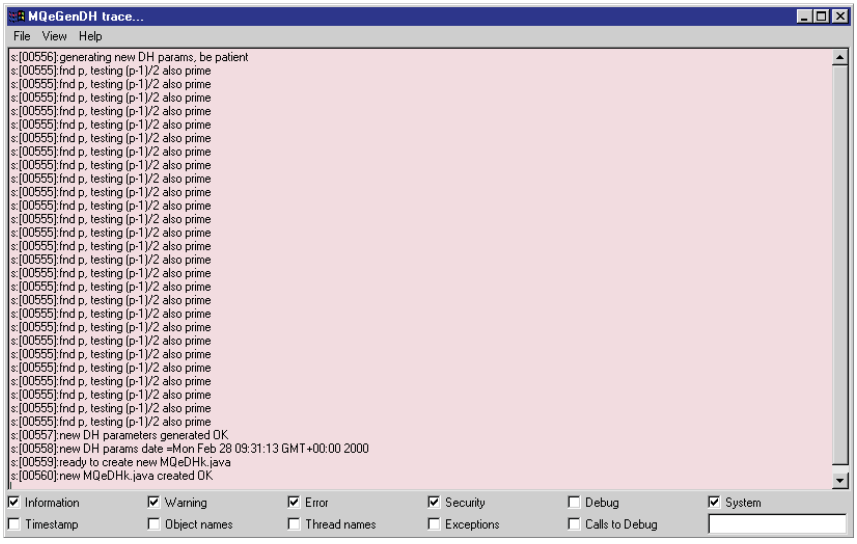
<parameter 2> 任意指定。トレース・クラス名 (たとえば、examples.awt.AwtMQeTrace) で、デフォルトではシステム・コンソールを使用。

戻り値 なし - 現行ディレクトリーで作成された新しい MQeDHk.java ファイル

例外 なし

例

```
java com.ibm.mqe.attributes.MQeGenDH 512 examples.awt.AwtMQeTrace
```



MQeGenDH genParams

構文

```
public void genParams(int length)
```

説明 ユーティリティーを呼び出して新しい DH 鍵ペアを生成し、新しい MQeDHk.java ファイルを作成します。

パラメーター

length パラメーターのビット長

戻り値 なし - 現行ディレクトリーで作成された新しい MQeDHk.java ファイル

例外 なし

MQeListCertificates

このクラスは、レジストリー内の共通証明書をリストするときに使用します。

パッケージ **com.ibm.mqe.attributes**

このクラスはオブジェクトの下位クラスです。

- コンストラクター
- メソッド

コンストラクター

MQeListCertificates

構文

1. `public MQeListcertificates()`
2. `public MQeListCertificates(string name,`
   `MQeFields regParams ) throws MQeException`

説明 MQeListCertificates オブジェクトを構成します。このコンストラクターには次の 2 つの形式があります。

1. 活動化メソッドの呼び出しに設定される属性を必要とするオブジェクトを作成します。
2. オブジェクトを作成して、活動化メソッドを自動的に呼び出します。

パラメーター

name このレジストリーに関連付けられた名前

regParams レジストリーの初期設定パラメーターを含む MQeFields オブジェクト

MQeRegistry.LocalRegType (ascii)
これにより、オープンするレジストリーのタイプが決まります。レジストリーが専用である場合、このパラメーターは `com.ibm.mqe.registry.MQePrivateSession` に設定する必要があります。レジストリーが公開である場合は、 `com.ibm.mqe.registry.MQeFileSession` に設定する必要があります。

MQeRegistry.DirName (ascii)
レジストリー・ファイルを入れておくディレクトリーの名前

MQeRegistry.PIN (ascii)
専用レジストリーの PIN。

MQeRegistry.Separator (ascii)

特定項目名のコンポーネント間で区切り文字として使う文字 (たとえば、<QueueManager><Separator><Queue>)。

これはストリングとして指定されますが、1 つの文字だけしか含められません。複数の文字を含めると、最初の文字だけが使われます。

レジストリーをオープンするたびに同じ区切り文字を使うようにします。レジストリーの使用を開始して項目を含めたら、区切り文字を変更してはなりません。

この値を指定しないと、デフォルトの + が使われます。

このパラメーターは、MQeNode_PublicRegistry という名前の公開レジストリーの場合は、ヌルにすることができます。そうでない場合はヌル以外でなければなりません。

戻り値 なし

例外

MQeException

Except_NotFound (値が指定されていなければならないのに、レジストリー・パラメーターがヌルの場合)。

MQe.Except_Data (レジストリー・パラメーターが誤りである場合)。

その他の値 (レジストリーをオープンするときにエラーが発生した場合)。

例

```
MQeListCertificates list1;
list1 = new MQeListCertificates();
try
{
MQeListCertificates list2;
MQeFields parms = new MQeFields();
Parms.putAscii(MQeRegistry.DirName, "Registry_Dir");
...
list2 = new MQeListCertificates("Reg2", parms);
}
catch (Exception e)
{...}
```

メソッド

メソッド	目的
activate	クラスを初期設定し、レジストリーをオープンします。
close	レジストリーをクローズして整理します。
getIssuer	証明書から issuer フィールドを戻します。
getNotAfter	証明書から valid-not-after フィールドを戻します。
getNotBefore	証明書から valid-not-before フィールドを戻します。
getSubject	証明書から subject フィールドを戻します。
getWTLSCertificate	レジストリー項目から証明書を戻します。
isNewCertificate	証明書が新しいフォーマットになっているかどうかを調べます。
readAllEntries	レジストリー内のすべての証明書の項目を読み取ります。
readEntry	レジストリー内の特定の証明書項目を読み取ります。

MQeListCertificates activate

構文

```
public void activate( String name,
                      MQeFields regParams ) throws MQeException
```

説明 クラスを初期設定し、レジストリーをオープンします。

パラメーター

name このレジストリーに関連付けられた名前

regParams レジストリーの初期設定パラメーターを含む MQeFields オブジェクト

MQeRegistry.LocalRegType (ascii)
これにより、オープンするレジストリーのタイプが決まります。レジストリーが専用である場合、このパラメーターは `com.ibm.mqe.registry.MQePrivateSession` に設定する必要があります。レジストリーが公開である場合は、 `com.ibm.mqe.registry.MQeFileSession` に設定する必要があります。

MQeRegistry.DirName (ascii)
レジストリー・ファイルを入れておくディレクトリーの名前

MQeRegistry.PIN (ascii)
専用レジストリーの PIN。

MQeRegistry.Separator (ascii)

特定項目名のコンポーネント間で区切り文字として使う文字 (たとえば、<QueueManager><Separator><Queue>)。

これはストリングとして指定されますが、1 つの文字だけしか含められません。複数の文字を含めると、最初の文字だけが使われます。

レジストリーをオープンするたびに同じ区切り文字を使うようにします。レジストリーの使用を開始して項目を含めたら、区切り文字を変更してはなりません。

この値を指定しないと、デフォルトの + が使われます。

このパラメーターは、 MQeNode_PublicRegistry という名前の公開レジストリーの場合は、ヌルにすることができます。そうでない場合はヌル以外でなければなりません。

戻り値 なし

例外

MQeException

Except_NotFound (値が指定されていなければならないのに、レジストリー・パラメーターがヌルの場合)。

MQe.Except_Data (レジストリー・パラメーターが誤りである場合)。

その他の値 (レジストリーをオープンするときにエラーが発生した場合)。

例

```
try
    {
    MQeListCertificates list1;
    MQeFields parms = new MQeFields();
    Parms.putAscii(MQeRegistry.DirName, "Registry_Dir");
   ...
    list1 = new MQeListCertificates();
    list1.activate("Reg1", parms);
    }
catch (Exception e)
    {
   ...
    }
```

MQeListCertificates readAllEntries

構文

```
public MQeFields readAllEntries() throws MQeException
```

説明 証明書のすべてのレジストリー項目を読み取り、それらの項目を MQeFields オブジェクトに戻します。

パラメーター

なし

戻り値 レジストリー内の証明書ごとに 1 つのフィールドを含む MQeFields オブジェクト。フィールドの名前は証明書の名前であり、フィールドの値そのものが証明書を含んだ MQeFields オブジェクトです。たとえば、レジストリーに 2 つの証明書が含まれている場合、 MQeFields オブジェクトには 2 つのフィールドがあり、それぞれのフィールドは MQeFields オブジェクトで、証明書が含まれています。

例外

MQeFields	Except_Closed (クラスがアクティブになっていない場合)。
	その他の値 (レジストリーの例を読み取るときにエラーが発生した場合)。

例

```
try
    {
    MQeListCertificates list1;
    MQeFields parms = new MQeFields();
    Parms.putAscii(MQeRegistry.DirName, "Registry_Dir");
   ...
    list1 = new MQeListCertificates("Reg1", parms);
    MQeFields certificates = readAllEntries();
    }
catch (Exception e)
    {
    ...
    }
```

MQeListCertificates readEntry

構文

```
public MQeFields readEntry(String certName) throws MQeException
```

説明 レジストリー内の特定の証明書項目を読み取り、その項目を MQeFields オブジェクトに戻します。

パラメーター

certName	読み取る証明書の名前。

戻り値

証明書のレジストリー項目を含んだ MQeFields オブジェクト。このオブジェクトは、**getWTLSCertificate()** メソッドに渡すことができます。

レジストリーに証明書がない場合には、ヌルが戻されます。

例外

MQeFields	Except_Closed (クラスがアクティブになっていない場合)。 その他の値 (レジストリーを読み取るときにエラーが発生した場合)。

例

```
try
   {
   MQeListCertificates list1;
   MQeFields parms = new MQeFields();
   Parms.putAscii(MQeRegistry.DirName, "Registry_Dir");
  ...
   list1 = new MQeListCertificates("Reg1", parms);
   MQeFields certificate = readEntry("Reg1");
   }
catch (Exception e)
   {
   ...
   }
```

MQeListCertificates getWTLSCertificate

構文

```
public Object getWTLSCertificate(MQeFields entry)
```

説明 レジストリー項目から証明書を戻します。

パラメーター

entry	レジストリー項目を含む MQeFields オブジェクト。これは、**readAllEntries()** によって戻された組み込みフィールド・オブジェクトのいずれかであるか、 **readEntry()** によって戻されたフィールド・オブジェクトである可能性があります。

戻り値

レジストリー項目内の証明書を表すオブジェクト。このオブジェクトは、**isNewCertificate()**、 **getSubject()**、**getIssuer()**、 **getNotBefore()**、および **getNotAfter()** に渡すことができます。

レジストリー項目に証明書が含まれていない場合は、ヌルが戻されます。

例外 なし

例

```
try
   {
   list1 = new MQeListCertificates("Reg1", parms);
   MQeFields certificates = readAllEntries();
   Enumeration enum = certificates.fields();
       while (enum.hasMoreElements())
        {
          // get the name of the certificate
          String entity = (String)enum.nextElement();
          // get the certificate's registry entry
          MQeFields certEntry = certificates.getFields( entity );
    // get the certificate object
    Object cert = getWTLSCertificate(certEntry);
        }
   }
catch (Exception e)
   {
   ...
   }
```

MQeListCertificates isNewCertificate

構文

```
public boolean isNewCertificate(Object certificate)
```

説明 証明書のフォーマットを調べます。古いフォーマットの証明書は、 MQSeries Everyplace バージョン 1.0 および 1.1 でサポートされていました。新しいフォーマットの証明書は、MQSeries Everyplace バージョン 1.2 で導入されました。

パラメーター

certificate **getWTLSCertificate()** で戻されるような、証明書を表すオブジェクト。

戻り値 このメソッドは、オブジェクトが新しい形式 (MQSeries Everyplace バージョン 1.2) の証明書である場合には、`true` を戻します。オブジェクトがそれ以外のものである場合は (証明書を表していなくても) `false` を戻します。

例外 なし

例

```
try
   {
   list1 = new MQeListCertificates("Reg1", parms);
   MQeFields certificates = readAllEntries();
   Enumeration enum = certificates.fields();
       while (enum.hasMoreElements())
        {
          // get the name of the certificate
          String entity = (String)enum.nextElement();
```

```
                    // get the certificate's registry entry
                    MQeFields certEntry = certificates.getFields( entity );
                    // get the certificate object
                    Object cert = getWTLSCertificate(certEntry);
                    if (isNewCertificate(cert))
                        System.out.println("new style certificate");
                    else
                        System.out.println("old style certificate");
                  }
            }
        catch (Exception e)
            {
            ...
            }
```

MQeListCertificates getSubject

構文

```
public String getSubject(Object certificate)
```

説明 証明書から *subject* ストリングを戻します。

パラメーター

certificate **getWTLSCertificate()** で戻されるような、証明書を表すオブジェクト。

戻り値 このメソッドは、証明書から *subject* フィールドを含むストリングを戻します。パラメーターが無効な証明書オブジェクトである場合は、ヌルが戻されます。

例外 なし

例

```
try
    {
    list1 = new MQeListCertificates("Reg1", parms);
    MQeFields certificates = readAllEntries();
    Enumeration enum = certificates.fields();
       while (enum.hasMoreElements())
          {
          // get the name of the certificate
          String entity = (String)enum.nextElement();
          // get the certificate's registry entry
          MQeFields certEntry = certificates.getFields( entity );
          // get the certificate object
          Object cert = getWTLSCertificate(certEntry);
          String subject = getSubject(cert);
          System.out.println("certificate " + entity + " subject is " + subject);
          }
    }
catch (Exception e)
    {
    ...
    }
```

MQeListCertificates getIssuer

構文

```
public String getIssuer(Object certificate)
```

説明 証明書から *issuer* ストリングを戻します。

パラメーター

certificate **getWTLSCertificate()** で戻されるような、証明書を表すオブジェクト。

戻り値 このメソッドは、証明書から *issuer* フィールドを含むストリングを戻します。パラメーターが無効な証明書オブジェクトである場合は、ヌルが戻されます。

例外 なし

例

```
try
    {
    list1 = new MQeListCertificates("Reg1", parms);
    MQeFields certificates = readAllEntries();
    Enumeration enum = certificates.fields();
       while (enum.hasMoreElements())
          {
          // get the name of the certificate
          String entity = (String)enum.nextElement();
          // get the certificate's registry entry
          MQeFields certEntry = certificates.getFields( entity );
          // get the certificate object
          Object cert = getWTLSCertificate(certEntry);
          String issuer = getSubject(cert);
          System.out.println("certificate " + entity + " issuer is " + issuer);
          }
    }
catch (Exception e)
     {
     ...
     }
```

MQeListCertificates getNotBefore

構文

```
public long  getNotBefore(Object certificate)
```

説明 証明書から *not before* 日付を戻します。これは、それ以前は証明書が無効である日付です。

パラメーター

certificate **getWTLSCertificate()** で戻されるような、証明書を表すオブジェクト。

戻り値

それ以前は証明書が無効である日付。日付は、1970 年 1 月 1 日の真夜中以降で、秒数を含んだ long で戻されます (日時は標準 UNIX 形式)。

日付の検索に誤りがある場合は、-1 が戻されます。

例外 なし

例

```
try
    {
    list1 = new MQeListCertificates("MQeNode_PublicRegistry", null);
    MQeFields certEntry = readEntry("myCert");
    // get the certificate object
    Object cert = getWTLSCertificate(certEntry);
    long notBefore = getNotBefore(cert);
    System.out.println("certificate invalid before " + new Date(notBefore * 1000));
    }
catch (Exception e)
    {
    ...
    }
```

MQeListCertificates getNotAfter

構文

```
public long  getNotAfter(Object certificate)
```

説明 証明書から *not after* 日付を戻します。これは、それ以後は証明書が無効になる日付です。

パラメーター

certificate **getWTLSCertificate()** で戻されるような、証明書を表すオブジェクト。

戻り値

それ以後は証明書が無効になる日付。日付は、1970 年 1 月 1 日の真夜中以降で、秒数を含んだ long で戻されます (日時は標準 UNIX 形式)。

日付の検索に誤りがある場合は、-1 が戻されます。

例外 なし

例

```
try
    {
    list1 = new MQeListCertificates("MQeNode_PublicRegistry", null);
    MQeFields certEntry = readEntry("myCert");
    // get the certificate object
    Object cert = getWTLSCertificate(certEntry);
    long notAfter = getNotBefore(cert);
    System.out.println("certificate invalid after " + new Date(notAfter * 1000));
    }
catch (Exception e)
    {
    ...
    }
```

MQeListCertificates close

構文

```
public void close()
```

説明 レジストリーをクローズして、リソースを解放します。

パラメーター

なし

戻り値 なし

例外 なし

例

```
try
    {
    list1 = new MQeListCertificates("MQeNode_PublicRegistry", null);
    MQeFields certEntry = readEntry("myCert");
    // get the certificate object
    Object cert = getWTLSCertificate(certEntry);
    long notBefore = getNotBefore(cert);
    System.out.println("certificate invalid before " + new Date(notBefore * 1000));
    list1.close();
    }
catch (Exception e)
    {
    ...
    }
```

MQeLocalSecure

このクラスは、LocalSecure オブジェクトを作成するときに使います。このオブジェクトには、使用しているアプリケーションで、所定の属性 (暗号化および圧縮) コンポーネントを適用してローカル・データを保護できるようにする、簡単なローカル・セキュリティー・サービスが備わっています。

パッケージ **com.ibm.mqe.attributes**

このクラスは MQe の下位クラスです。

- コンストラクター
- メソッド

コンストラクター

MQeLocalSecure

構文

```
public MQeLocalSecure( )
```

説明 MQeLocalSecure オブジェクトを構成します。

パラメーター

なし

戻り値 なし

例外 なし

例

```
MQeLocalSecure ls = new MQeLocalSecure( );
```

関連する関数

- MQeAttribute

メソッド

メソッド	目的
open	使用しているアプリケーションで、ターゲット・ファイルを識別できるようにします。
read	ターゲット・ファイルのデータを読み取り、保護を解除し、戻します。
write	所定のデータをターゲット・ファイルへ保護し書き込みます。

MQeLocalSecure open

構文

```
public void open( String  directory,
                  Object  fileName )
```

説明 ターゲット・ファイル名を設定します。

パラメーター

directory ターゲット・ファイル・ディレクトリーを示すストリング

fileName ターゲット・ファイル名を示すストリング

戻り値 なし

例外 なし

関連する関数

- **write()**
- **read()**

MQeLocalSecure read

構文

```
public byte[] read( MQeAttribute attr,
                    String localCipherKey ) throws Exception
```

説明 所定のターゲット・ファイル名のデータを読み取り、保護を解除します。

パラメーター

attr データの保護を解除するときに適用する MQeAttribute

localCipherKey データの保護を解除するときに使うパスワード (パスフレーズ) のストリング

戻り値 なし

例外

MQeException	Except_S_InvalidAttribute, ″no cryptor″
	Except_S_InvalidAttribute, ″illegal cryptor″
	Except_S_InvalidAttribute, ″illegal authenticator or compressor″
	MQe.Except_Data, ″wrong cipher″
例外	java.io

例

```
try
    {
    MQeDESCryptor des  = new MQeDESCryptor( );
    MQeAttribute desA  = new MQeAttribute( null, des, null);
```

```
    MQeLocalSecure ls  = new MQeLocalSecure( );
    ls.open( ".¥¥", "TestSecureData.txt" );
    String outData     = byteToAscii( ls.read( A,
                                       "It_is_a_secret" ) );
    Trace ( "i: unprotected data = " + outData);
    ...
    }
  catch ( Exception e )
    {
    ...
    }
```

MQeLocalSecure write

構文

```
public void write ( byte[] data,
                    MQeAttribute attr,
                    String localCipherKey) throws Exception
```

説明 データを保護し、所定のターゲット・ファイル名に書き込みます。

パラメーター

data	保護するデータ
attr	データを保護するときに適用する MQeAttribute
localCipherKey	データを保護するときに使うパスワード (パスフレーズ) のストリング

戻り値 なし

例外

MQeException	Except_S_InvalidAttribute, ″no cryptor″
	Except_S_InvalidAttribute , ″illegal cryptor″
	Except_S_InvalidAttribute, ″illegal authenticator or compressor″
例外	java.io

例

```
try
    {
    MQeDESCryptor des  = new MQeDESCryptor( );
    MQeAttribute desA  = new MQeAttribute( null, des, null);
    MQeLocalSecure ls  = new MQeLocalSecure( );
    ls.open( ".¥¥", "TestSecureData.txt" );
    ls.write( asciiToByte( "0123456789abcdef..." ),
                                  desA, "It_is_a_secret" );
    ...
    }
```

```
catch ( Exception e )
  {
  ...
  }
```

MQeLZWCompressor

このクラスは、LZWCompressor オブジェクトを作成するときに使います。このオブジェクトは、属性オブジェクトによって使われるとき、その属性オブジェクトに LZW 圧縮を実行するメカニズムを提供します。属性オブジェクトは、チャネルおよび MQeFields オブジェクトに関連付けられています。

パッケージ **com.ibm.mqe.attributes**

このクラスは、MQeCompressor の下位クラスです。

コンストラクター

MQeLZWCompressor

構文

```
public MQeLZWCompressor( )
```

説明 MQeLZWCompressor オブジェクトを構成します。

パラメーター

なし

戻り値 なし

例外 なし

例

```
try
    {
    MQeLZWCompressor lzw  = new MQeLZWCompressor();
    MQeAttribute lzwA     = new MQeAttribute(null, null, lzw);
    ...
    }
  catch ( Exception e )
    {
    ...
    }
```

関連する関数

- MQeCompressor
- MQeAttribute
- MQeLocalSecure
- MQeMAttribute
- MQeMTrustAttribute

MQeMARSCryptor

このクラスは、MARSCryptor オブジェクトを作成するときに使います。このオブジェクトは、属性オブジェクトによって使われるときに、 MARS 暗号化を実行するメカニズムを提供します。属性オブジェクトは、チャネルおよび MQeFields オブジェクトに関連付けられています。

パッケージ **com.ibm.mqe.attributes**

このクラスは、MQeCryptor の下位クラスです。

コンストラクター

MQeMARSCryptor

構文

```
public MQeMARSCryptor( )
```

説明 MQeMARS 暗号機能オブジェクトを構成します。

パラメーター

なし

戻り値 なし

例外 なし

例

```
try
    {
    MQeMARSCryptor mars   = new MQeMARSCryptor();
    MQeAttribute marsA    = new MQeAttribute(null, mars, null);
    ...
    }
  catch ( Exception e )
    {
    ...
    }
```

関連する関数

- MQeCryptor
- MQeAttribute
- MQeLocalSecure
- MQeMAttribute
- MQeMTrustAttribute

MQeMAttribute

このクラスは、メッセージに添付されるときに簡単なメッセージ・レベルの保護を可能にする属性オブジェクトを作る場合に使います。

パッケージ **com.ibm.mqe.attributes**

このクラスは、MQeAttribute の下位クラスです。

コンストラクター

MQeMAttribute

構文

```
public MQeMAttribute( MQeAuthenticator authenticator,
MQeCryptor    cryptor,
MQeCompressor    compressor) throws Exception
```

説明 MQeMAttribute オブジェクトを構成します。

パラメーター

authenticator ヌルまたは MQeAuthenticator オブジェクトへのオブジェクト参照

cryptor シンメトリック MQeCryptor オブジェクト (MQeDESCryptor、 MQe3DESCryptor、 MQeRC4Cryptor、 MQeRC6Cryptor または MQeMARSCryptor) へのオブジェクト参照

compressor ヌルまたは MQeCompressor オブジェクト (MQeRleCompressor または MQeLZWCompressor) へのオブジェクト参照

戻り値 なし

例外

MQeException

Except_ Not Supported, ″invalid authenticator″

Except_ Not Supported, ″invalid cryptor″

Except_ Not Supported, ″invalid compressor″

例

```
class MySampleClass extends MQe
  {
  /* application on initiating QueueManager:                    */
  /* -prepare to use MQeMAttribute with Rle Compressor           */
  /*  and DES Cryptor with key = It_is_a_secret                  */
```

```
  MQeKey localkey           = new MQeKey();
  localkey.setLocalKey( "It_is_a_secret");
  MQeDESCryptor des         = new MQeDESCryptor();
  MqeRleCompressor rle      = new MQeRleCompressor();
  MQeMAttribute protMAttr   = new MQeMAttribute( null, des, rle );
  protMAttr.setKey( localkey );
  /* construct Message and protect with the MQeMAttribute             */
  MQeMessageObj MsgObj      = new MQeMessageObject();
  MsgObj.setAttribute( protMAttr );         /* add test message data  */
  MsgObj.putAscii("MsgData", "0123456789abcdef....");
  trace ("i: input message data = " + MsgObj.getAscii("MsgData") );
  /* assume MQeQueueManager instance initQM started, PutMessage        */
  initQM.putMessage( targetQMgrName, targetQName, MsgObj ,null, 0);
...
...//
...//.
...//.
  /* application on recipient QueueManager:                            */
  /* -prepare to use MQeMAttribute with key = It_is_a_secret           */
  MQeKey localkey           = new MQeKey();
  localkey.setLocalKey( "It_is_a_secret");
  MQeDESCryptor des         = new MQeDESCryptor();
  MqeRleCompressor rle      = new MQeRleCompressor();
  MQeMAttribute protMAttr   = new MQeMAttribute( null, des, rle );
  protMAttr.setKey( localkey );
  /* assume MQeQueueManager instance recipQM started, GetMessage      */
  MQeMsgObject MsgObj       = recipQM.getMessage(thisQMgrName,
                                 thisQName, null, protMAttr, 0);
  trace ("i: output message data = " + MsgObj.getAscii( "MsgData" ) );


  }
```

関連する関数

MQeAttribute

メソッド

メソッド	目的
decodeData	提供されたデータを復号または圧縮解除 (またはその両方) します。
encodeData	提供されたデータを暗号化または圧縮 (またはその両方) します。
setKey	キーと属性を関連付けます。

MQeMAttribute decodeData

構文

```
public byte[] decodeData( MQeChannel channel,
                                    byte data[],
                                    int offset,
                                    int count ) throws Exception
```

説明 *data*、 *offset* および長さ *count* で示されるバイトをデコードする (復号または圧縮解除する (またはその両方)) ときに呼び出されます。

注: このメソッドは内部での使用のためのものであり、通常は、アプリケーションによって呼び出されることはありません。

パラメーター

channel	ヌル (使いません)
data	デコードするデータを含むバイト配列へのオブジェクト参照
offset	*data* 配列での開始バイトを指定する整数索引
count	デコードするバイト数を示す整数カウント

戻り値 デコードされたデータ

例外

MQeException	Except_Data, ″data tampering detected″
	Except_S_NoPresetKeyAvailable

MQeMAttribute encodeData

構文

```
public byte[] encodeData( MQeChannel channel,
                                    byte data[],
                                    int offset,
                                    int count ) throws Exception
```

説明 *data*、*offset* および長さ *count* で示されるバイトをエンコードする (暗号化または圧縮 (またはその両方)) します。

注: このメソッドは内部での使用のためのものであり、通常は、アプリケーションによって呼び出されることはありません。

パラメーター

channel	ヌル (使いません)
data	エンコードするデータを含むバイト配列へのオブジェクト参照
offset	*data* 配列での開始バイトを指定する整数索引
count	エンコードするバイト数を示す整数カウント

戻り値 なし

例外

MQeException	Except_ Not Supported, ″invalid cryptor″

Except_ Not Supported, ″invalid compressor″

Except_S_NoPresetKeyAvailable

MQeMAttribute setKey

構文

```
public void setKey(MQeKey key)
```

説明 このメソッドは、キーと属性を関連付けます。属性が暗号機能を持っている場合に、必要になります。

パラメーター

キー 属性の暗号機能と共に使うキー・オブジェクト

戻り値 なし

例外

MQeException NotAllowed (キーがすでに設定されている場合)。

MQeMTrustAttribute

このクラスは、メッセージ・オブジェクトに対するメッセージ・レベルの保護を可能にする属性オブジェクトを作るときに使います。これは、以下の方法で行うことができます。

- 発信元の (ISO9796) ディジタル署名を妥当性検査することにより、受信側がメッセージの起点を突き止められるようにします (拒否はなし)
- 属性の暗号機能を使ってメッセージの秘密性が保護されます
- メッセージの整合性が妥当性検査されます
- 意図した受信側はメッセージ・データを復元することだけが可能です

パッケージ **com.ibm.mqe.attributes**

このクラスは、MQeAdminQueueAdminMsg の下位クラスです。

コンストラクター

MQeMTrustAttribute

このコンストラクターには次の 2 つの形式があります。

1. 活動化メソッドの呼び出しに設定される属性を必要とするオブジェクトを作成します。
2. オブジェクトを作成して、活動化メソッドを自動的に呼び出します。

構文

```
1. public MQeMTrustAttribute( )
2.
   public MQeMTrustAttribute( MQeAuthenticator authenticator,
                              MQeCryptor cryptor,
                              MQeCompressor compressor) throws Exception
```

説明 MQeMTrustAttribute オブジェクトを構成します。

パラメーター

authenticator ヌル (使いません)

cryptor シンメトリック MQeCryptor オブジェクト (MQeDESCryptor、 MQe3DESCryptor、 MQeRC4Cryptor、 MQeRC6Cryptor または MQeMARSCryptor) へのオブジェクト参照

compressor ヌル (使いません)

戻り値 なし

例外

MQeException Except_ Not Supported, ″invalid cryptor″

例

```
class MySampleClass extends MQe
  {
  /* application on initiating QueueManager:                          */
  /* use MQeMTrustAttribute to protect a message between              */
  /* pre-reg'd initiator 'Bruce1' and recipient 'Bruce8'              */
  /* assume initiator's QueueManager initQM started                   */
  /* setup MTrustAttribute                                            */
  MQeMARSCryptor mars          = new MQeMARSCryptor( );
  MQeMTrustAttribute msgA = new MQeMTrustAttribute( null, mars, null );

  /* setup instantiate & activate sender (Bruce1) PrivReg             */
  String EntityName            = "Bruce1";
  String EntityPIN             = "12345678";
  Object KeyRingPassword       = "It_is_a_secret";
  MQePrivateRegistry sendreg   = new MQePrivateRegistry( );
  sendreg.activate( EntityName, ".//MQeNode_PrivateRegistry",
                    EntityPIN,KeyRingPassword, null, null );

  /* set target entity's registry name into sender's reg              */
  sendreg.setTargetRegistryName("Bruce8");
  /* set MTrustAttribute s PrivateRegistry = sendreg                  */
  msgA.setPrivateRegistry( sendreg );
  /* instantiate and activate Public Registry which has               */
  /* (or gets) MiniCert of intended recipient (Bruce8)                */
  MQePublicRegistry pr         = new MQePublicRegistry( );
  pr.activate( "MQeNode_PublicRegistry", ".//" );
  /* set MTrustAttribute's PublicRegistry & HomeServer                */
  msgA.setPublicRegistry( pr);
  msgA.setHomeServer( MyHomeServer + ":8082" );

  /* create message object and add some test data                     */
  MQeMsgObject msgObj      = new MQeMsgObject( );
  msgObj.putArrayOfByte( "TestData",
                      asciiToByte("0123456789abcdef....") );
  /* protect with MQeMTrustAttribute and PutMessage                   */
  msgObj.setAttribute( msgA );
  initQM.putMessage( targetQMgrName, targetQName, msgObj, null, 0);

  ...
  /* application on recipient QueueManager:                           */
  /* use MQeMTrustAttribute to recover the message from.              */
  /* pre-registered initiator 'Bruce1' and recipient 'Bruce8'         */
  /* assume recipient's QueueManager recipQM started                  */
  ...
  /* setup  MQeMTrustAttribute                                        */
  MQeMARSCryptor mars          = new MQeMARSCryptor( );
  MQeMTrustAttribute msgA
                = new MQeMTrustAttribute(null, mars, null);

  /* setup recipient's Private Registry                               */
  String EntityName            = "Bruce8";
  String EntityPIN             = "12345678";
  Object KeyRingPassword       = "It_is_a_secret";
```

```
/* instantiate and activate recipient's Private Registry          */
MQePrivateRegistry recipreg  = new MQePrivateRegistry( );
recipreg.activate( EntityName, ".//MQeNode_PrivateRegistry",
                   EntityPIN, KeyRingPassword, null, null );
/* set MTrustAttribute PrivateRegistry = recipreg                 */
msgA.setPrivateRegistry( recipreg );
/* instantiate and activate Public Registry which has             */
/* (or gets) MiniCert of originator (Bruce1)                      */
MQePublicRegistry pr         = new MQePublicRegistry( );
pr.activate( "MQeNode_PublicRegistry", ".//" );
/* set MTrustAttribute's PublicRegistry & HomeServer              */
msgA.setPublicRegistry( pr);
msgA.setHomeServer( MyHomeServer + ":8082" );
/* use MQeMTrustAttribute with GetMessage to recover msg          */
MQeMsgObject MsgObj = SvrQM.getMessage( TargetQMgrName,
    TargetQName,null, msgA, 0 );
trace("i: Data restored from MTrustAttr protected Msg ="
      + byteToAscii(MsgObj.getArrayOfByte("TestData") ) );
}
```

関連する関数

- MQePrivateRegistry
- MQePublicRegistry

メソッド

メソッド	目的
decodeData	提供されたデータをデコードします。
encodeData	提供されたデータをエンコードします。
setHomeServer	ホーム・サーバー / 代替 MQSeries Everyplace ノードのアドレスを設定します。
setPrivateRegistry	アクティブな専用レジストリーを設定します。
setPublicRegistry	アクティブな公開レジストリーを設定します。
setTarget	意図した受信側の名前を属性に追加します。

MQeMTrustAttribute decodeData

構文

```
public byte[] decodeData( MQeChannel channel,
                                    byte data[],
                                    int offset,
                                    int count ) throws Exception
```

説明 *data*、 *offset* および長さ *count* で示されるバイトをデコードする (復号または圧縮解除する (またはその両方)) ときに呼び出されます。

注: このメソッドは内部での使用のためのものであり、通常は、アプリケーションによって呼び出されることはありません。

パラメーター

channel	ヌル (使いません)
data	デコードするデータを含むバイト配列へのオブジェクト参照
offset	*data* 配列での開始バイトを指定する整数索引
count	デコードするバイト数を示す整数カウント

戻り値 デコードされたデータ

例外

MQeException	`Except_S_RegistryNotAvailable` ″intended recipient's PrivateRegistry not available″
	`Except_S_MiniCertNotAvailable` ″cannot recover data, sender's MiniCert not available″
	`Except_S_RegistryNotAvailable` ″cannot recover data target(recipient) PrivateRegistry not available″
	`Except_S_BadIntegrity`, ″validating data from < Sender> data tampering detected″
	`Except_S_InvalidSignature`, ″validating data from < Sender > bad signature″

MQeMTrustAttribute encodeData

構文

```
public byte[] encodeData( MQeChannel channel,
                                    byte data[],
                                    int offset,
                                    int count ) throws Exception
```

説明 *data*、**offset** および長さ *count* で示されるバイトをエンコードする (暗号化または圧縮 (またはその両方)) します。

注: このメソッドは内部での使用のためのものであり、通常は、アプリケーションによって呼び出されることはありません。

パラメーター

channel	ヌル (使いません)
data	エンコードするデータを含むバイト配列へのオブジェクト参照
offset	*data* 配列での開始バイトを指定する整数索引

count エンコードするバイト数を示す整数カウント

戻り値 なし

例外

MQeException Except_S_MiniCertNotAvailable, ″cannot protect data, target 最小限の証明 not available″

MQeMTrustAttribute setHomeServer

構文

```
public void setHomeServer( String homeServerAddrPort)
                           throws Exception
```

説明 MQeMTrustAttribute のホーム・サーバー・アドレスを設定するときに呼び出されます。メッセージを保護するときに使う場合、 **encodeData** は、アクティブな公開レジストリーから、意図した受信側の最小限の証明を取得しようとします。見付からなくてもホーム・サーバーのアドレスが設定されていれば、そのホーム・サーバーから最小限の証明を要求し、後で使うために、アクティブな公開レジストリーに保管します。メッセージの回復に使う場合は、 **decodeData** は、アクティブな公開レジストリーから、起動側の最小限の証明を取得しようとします。見付からなくてもホーム・サーバーのアドレスが設定されていれば、そのホーム・サーバーから最小限の証明を要求し、後で使うために、アクティブな公開レジストリーに保管します。

パラメーター

homeServerAddrPort 認証可能なエンティティーの最小限の証明群を含む公開レジストリーがある、別の MQeNode (たとえば、HomeServer) のアドレス。使われるフォーマットは tcpname:port か tcpaddress:port です。

戻り値 なし

例外

MQeException Except_NotAllowed, ″illegal SetPublicRegistry″

MQeMTrustAttribute setPrivateRegistry

構文

```
public void setPrivateRegistry( MQePrivateRegistry privreg)
                                throws Exception
```

説明 MQeMTrustAttribute のアクティブな専用レジストリーを設定するときに呼び出

されます。メッセージを保護するときに使う場合、これは送信側の専用レジストリーであり、メッセージを回復するときに使う場合は、受信側の専用レジストリーになります。

パラメーター

privreg 送信側または受信側の認証可能なエンティティーの、アクティブにされた MQePrivateRegistry。

戻り値 なし

例外

MQeException Except_NotAllowed, ″illegal SetPrivateRegistry″

MQeMTrustAttribute setPublicRegistry

構文

```
public void setPublicRegistry( MQePublicRegistry pubreg)
                                                  throws Exception
```

説明 MQeMTrustAttribute の公開レジストリーを設定するときに呼び出されます。メッセージを保護するときに使う場合、これは意図した受信側の最小限の証明を持つ (取得する) 公開レジストリーであり、メッセージを回復するときに使う場合、送信側の最小限の証明を持つ (取得する) 公開レジストリーになります。

パラメーター

pubreg 保護に使う場合、意図した受信側の最小限の証明を含み、回復に使う場合には、送信側の最小限の証明を含むアクティブにされた MQePublicRegistry。

戻り値 なし

例外

MQeException Except_NotAllowed, ″illegal SetPublicRegistry″

MQeMTrustAttribute setTarget

構文

```
public boolean setTarget(string target)
```

説明

このメソッドは、意図した受信側の名前を属性に追加します。メッセージを暗号化するために受信側の公開鍵を検索するときに使います。この名前はメッセージにも追加され、受信側の専用レジストリーを検索し、メッセージの暗号化を可能にするために宛先で使われます。

このメソッドは、クラス MQePrivateRegistry 内のメソッド **setTargetRegistryName()** に優先して使う必要があります。 **setTarget()** と **setTargetRegistryName()** の両方が呼び出された場合は、 **setTarget()** で指定された名前が使われます。

パラメーター

target 意図した受信側の名前が入ったストリング

戻り値 なし

例外

MQeException NotAllowed。名前がすでに設定されている場合に示されます。

MQeRC4Cryptor

このクラスは、RC4Ccryptor オブジェクトを作成するときに使います。このオブジェクトは、属性オブジェクトによって使われるときに、その属性オブジェクトに RC4 暗号化を実行するメカニズムを提供します。属性オブジェクトは、チャネルおよび MQeFields オブジェクトに関連付けられています。

パッケージ **com.ibm.mqe.attributes**

このクラスは、MQeCryptor の下位クラスです。

コンストラクター

MQeRC4Cryptor

構文

```
public MQeRC4Cryptor( )
```

説明 MQeRC4Cryptor オブジェクトを構成します。

パラメーター

なし

戻り値 なし

例外

MQeException	Except_S_Cipher, ″cipRC4, wrong cipher or key″

例

```
try
    {
    MqeRC4Cryptor rc4    = new MQeRC4Cryptor();
    MQeAttribute rc4A    = new MQeAttribute(null, rc4, null);
    ...
    }
  catch ( Exception e )
    {
    ...
    }
```

関連する関数

- MQeCryptor
- MQeAttribute
- MQeLocalSecure
- MQeMAttribute
- MQeMTrustAttribute

MQeRC6Cryptor

このクラスは、RC6Cryptor オブジェクトを作成するときに使います。このオブジェクトは、属性オブジェクトによって使われるときに、その属性オブジェクトに RC6 暗号化を実行するメカニズムを提供します。属性オブジェクトは、チャネルおよび MQeFields オブジェクトに関連付けられています。

パッケージ **com.ibm.mqe.attributes**

このクラスは、MQeCryptor の下位クラスです。

コンストラクター

MQeRC6Cryptor

構文

```
public MQeRC6Cryptor( )
```

説明 MQeRC6Cryptor オブジェクトを構成します。

パラメーター

なし

戻り値 なし

例外

MQeException	`Except_S_Cipher`, ″cipRC6, wrong cipher or key″

例

```
try
    {
    MQeRC6Cryptor rc6  = new MQeRC6Cryptor();
    MQeAttribute  rc6A = new MQeAttribute(null, rc6, null);
    ...    }
  catch ( Exception e )
    {
    ...
    }
```

関連する関数

- MQeCryptor
- MQeAttribute
- MQeLocalSecure
- MQeMAttribute
- MQeMTrustAttribute

MQeRleCompressor

このクラスは、RleCompressor オブジェクトを作成するときに使います。このオブジェクトは、属性オブジェクトによって使われるときに、その属性オブジェクトに Rle 圧縮を実行するメカニズムを提供します。属性オブジェクトは、チャネルおよび MQeFields オブジェクトに関連付けられています。

パッケージ　**com.ibm.mqe.attributes**

このクラスは、MQeCompressor の下位クラスです。

コンストラクター

MQeRleCompressor

構文

```
public MQeRleCompressor( )
```

説明　MQeRleCompressor オブジェクトを構成します。

パラメーター

なし

戻り値　なし

例外　なし

例

```
try
    {
    MQeRleCompressor rle  = new MQeRleCompressor();
    MQeAttribute rleA     = new MQeAttribute(null, null, rle);
    ...
    }
  catch ( Exception e )
    {
    ...
    }
```

関連する関数

- MQeCompressor
- MQeAttribute
- MQeLocalSecure
- MQeMAttribute
- MQeMTrustAttribute

MQeWTLSCertAuthenticator

このクラスは、WTLSCertAuthenticator オブジェクトを作成するときに使います。このオブジェクトは、属性オブジェクトによって使われるときに、その属性オブジェクトに相互認証に基づく最小限の証明を実行するメカニズムを提供します。これは、チャネル・オブジェクトに関連付けられた MQeAttribute オブジェクトに適用されます。

パッケージ **com.ibm.mqe.attributes**

このクラスは、MQeAuthenticator の下位クラスです。

コンストラクター

MQeWTLSCertAuthenticator

構文

```
public MQeWTLSCertAuthenticator( )
```

説明 MQeWTLSCertAuthenticator オブジェクトを構成します。

パラメーター

なし

戻り値 なし

例外 なし

例

```
try
    {
    MQeWTLSCertAuthenticator wtls = new MQeWTLSCertAuthenticator( );
    MQeDESCryptor des             = new MQeDESCryptor( );
    MQeAttribute wtlsA            = new MQeAttribute(wtls, des, null);
    ...
    }
  catch ( Exception e )
    {
    ...
    }
```

関連する関数

- MQeAuthenticator
- MQeAttribute

MQeXORCryptor

このクラスは、XORCryptor オブジェクトを作成するときに使います。このオブジェクトは、属性オブジェクトによって使われるときに、その属性オブジェクトに XOR エンコードを実行するメカニズムを提供します。属性オブジェクトは、チャネルおよび MQeFields オブジェクトに関連付けられています。

パッケージ **com.ibm.mqe.attributes**

このクラスは、MQeCryptor の下位クラスです。

コンストラクター

MQeXORCryptor

構文

```
public MQeXORCryptor()
```

説明 MQeXORCryptor オブジェクトを構成します。

パラメーター

なし

戻り値 なし

例外 なし

例

```
try
    {
     MQeXorCryptor xor  = new MQeXorCryptor( );
     xor.setEncryptKey ( asciiToByte("It_is_a_secret") );
     NTAuthenticator nt = new NTAuthenticator( );
     String inData      = "0123456789abcdef...";
     trace("i: TestXOR, indata = " + inData);
     MQeFields tempf    = new MQeFields();
     tempf.putAscii( "testdata", inData);
     MQeAttribute attr1 = new MQeAttribute( );
     attr1.activate( null, nt, xor, null );
     tempf.setAttribute ( attr1 ) ;
     byte[] temp = tempf.dump();

     //

     MQeFields tempf2   = new MQeFields();
     MQeXorCryptor xor2 = new MQeXorCryptor( );
     xor2.setDecryptKey ( asciiToByte("It_is_a_secret") );
     NTAuthenticator nt2 = new NTAuthenticator( );
     MQeAttribute attr2 = new MQeAttribute( );
     attr2.activate( null, nt2, xor2, null );
     tempf2.setAttribute ( attr2 );
     tempf2.restore( temp );
```

```
    trace("i: TestXORSecure, outdata = " + tempf2.getAscii("testdata"));
   }
 catch ( Exception e )
   {
   //
   }
```

関連する関数

- MQeCryptor
- MQeAttribute

メソッド

メソッド	目的
setDecryptKey	暗号機能の復号鍵を明示的に設定します。
setEncryptKey	暗号機能の暗号化鍵を明示的に設定します。

MQeXORCryptor setDecryptKey

構文

```
public void setDecryptKey ( Object newKey) throws Exception
```

説明 暗号機能の復号鍵を明示的に設定します。

パラメーター

newKey 暗号機能の復号鍵が派生するときの byte[] シード値。

戻り値 なし

例外 なし

MQeXORCryptor setEncryptKey

構文

```
public void setEncryptKey ( Object newKey) throws Exception
```

説明 暗号機能の暗号化鍵を明示的に設定します。

パラメーター

newKey 暗号機能の暗号化鍵が派生するときの byte[] シード値。

戻り値 なし

例外 なし

第5章 com.ibm.mqe.registry のクラス

この節には、以下の MQSeries Everyplace クラスについての詳細が載せられています。

表 15. パッケージ com.ibm.mqe.registry のクラス

クラス名	目的
MQePrivateRegistry	一群の専用および共通オブジェクトへの制御アクセスを可能にする、専用レジストリー・オブジェクトを作成します。
MQePrivateRegistryConfigure	専用レジストリーを構成するときに使います。
MQePublicRegistry	一群の専用および共通オブジェクトへの制御アクセスを可能にする、公開レジストリー・オブジェクトを作成します。

MQePrivateRegistry

このクラスは、MQePrivateRegistry オブジェクトを作成するときに使います。 MQePrivateRegistry クラスは、 MQeRegistry の下位クラスで、一群の専用および共通オブジェクト (たとえば、証明書) への制御アクセスを可能にします。 MQePrivateRegistry オブジェクトもディジタル署名と復号サービスをサポートしており、レジストリーの専用オブジェクト (たとえば、認証可能なエンティティーの秘密鍵) を内部的に使うことができるので、専用レジストリーに入れておきます。

パッケージ **com.ibm.mqe.registry**

このクラスは、MQeRegistry の下位クラスです。

- コンストラクター
- メソッド

コンストラクター

MQePrivateRegistry

構文

```
public MQePrivateRegistry( )
```

説明 MQePrivateRegistry オブジェクトを構成します。

パラメーター

なし

戻り値 なし

例外 なし

関連する関数

- MQePublicRegistry

メソッドの要約

メソッド	目的
activate	MQePrivateRegistry インスタンスをオープンしてアクティブにします。
deleteCertificate	証明書所有者の最小限の証明を削除します。
getCertificate	証明書所有者の最小限の証明を戻します。
getRegistryName	専用レジストリーの認証可能なエンティティー名を取得します。
resetPIN	専用アクセスを制御する PIN をリセットします。
setTargetRegistryName	意図した受信側 (認証可能なエンティティー) の専用レジストリーの名前を設定します。

MQePrivateRegistry activate

構文

```
public void activate (String entityName,
                      String dirName,
                      String pin,
                      Object keyRingPassword,
                      Object certReqPIN,
                      Object caIPAddrPort ) throws Exception
```

説明

この *entityName* がある専用レジストリーが存在する場合、 **activate()** は、指定した *pin* を使い、専用レジストリーをオープンしようとします。存在しない場合、**activate()** は、新しい専用レジストリーを作成してオープンし、指定した *pin* でアクセスできるようにします。

ヌル以外の最小限の証明サーバー・アドレス (*caIPAddrPort*) が指定される場合、 **activate()** は、その専用レジストリーを調べ、その所有者がすでに登録されている (つまり、独自の最小限の証明を持っている) かどうかを見極めます。まだ登録されていなければ (最小限の証明がない)、 **activate()** は自動登録を実行します。これにより、*entityName* が自動登録され、以下の作業が実行されます。

- 所有する *entityName* のために新しい RSA 鍵ペアを生成する
- 所定の *keyRingPassword* の派生物を使って保護した後で、秘密鍵 (CRTKey) を専用レジストリーへ保管する
- *newCertificateRequest* の公開鍵を、指定された最小限の証明サーバー・アドレスへパッケージし、 *entityName* が付けられた要求と指定された (事前割り振り) 最小限の証明要求 pin (*certReqPIN*) を識別する
- 発行された最小限の証明を専用レジストリーに保管してから、**getCertificate** 要求を送信し、最小限の証明サーバーの (所有する) 最小限の証明を取得して使用レジストリーへ保管する

パラメーター

entityName PrivateRegistry 所有者の EntityName

dirName PrivateRegistry へのパス

pin 専用レジストリーをオープンするために使う番号、パスワード (パスフレーズ)

keyRingPassword
エンティティーの秘密鍵を保護するために使うストリング・パスワード (パスフレーズ)

certReqPIN 自動登録できるように、最小限の証明サーバーの管理者によ

ってエンティティーに事前割り振りされた、一回だけ使用する*認証要求番号* のあるストリング

caIPAddrPort ソリューションの最小限の証明サーバーの TCP アドレスとポートが示されたストリング (たとえば、aname.hursley.ibm.com:8082)

戻り値 なし

例外

MQeException	Except_PrivateReg_BadPIN, ″Activating_EntityName_PrivateRegistry″
	Except_PrivateReg_ActivateFailed
	Except_PrivateReg_ActivateFailed, ″Registration exception ″

例

```
class MySampleClass extends MQe
  {
  try
    {
    /* setup Private Registry activate parameters                    */
    String entityName          = "Bruce";
    String dirName             = ".//" + EntityName;
    String entityPIN           = "12345678";
    Object keyRingPassword     = "It_is_a_secret";
    Object certReqPIN          = "12345678";
    Object caIPAddrPort        = "aname.hursley.ibm.com:8082";
    /* instantiate and activate a Private Registry...     */
    MQePrivateRegistry preg    = new MQePrivateRegistry( );
    /* instantiate and activate the Private Registry                 */
    preg.Activate( entityName,     /* name of entity owning privreg  */
                   dirName,        /* params to open file regsess'n  */
                   entityPIN,      /* Private Registry access PIN    */
                   keyRingPassword,/* pwd/phrase protecting CRTKey   */
                   certReqPIN,     /* prereg MiniCertSvr certreqPIN  */
                   caIPAddrPort);  /* trusted MiniCertSvr addr:port  */
    }
  catch ( Exception e )
    {
    }
```

関連する関数

MqeLocalSecure

MQePrivateRegistry deleteCertificate

構文

```
public MQeFields deleteCertificate( String certificateOwner )
                                                  throws MQeException
```

説明 証明書所有者の最小限の証明を削除します。

パラメーター

certificateOwner
専用レジストリー所有者の名前

戻り値 なし

例外

MQeException	`Except_Reg_DoesNotExist`, ″Entry does not exist″
	`Except_Reg_DeleteFailed`, ″Error deleting entry″

関連する関数

getCertificate()

MQePrivateRegistry getCertificate

構文

```
public MQeFields getCertificate( String certificateOwner )
                                                    throws MQeException
```

説明 証明書所有者の最小限の証明を戻します。

パラメーター

certificateOwner
専用レジストリー所有者の名前

戻り値 最小限の証明

例外

MQeException `Except_Reg_ReadFailed`, ″Error reading entry″

関連する関数

deleteCertificate()

MQePrivateRegistry getRegistryName

構文

```
public String getRegistryName( )
```

説明 所有するエンティティー名を戻します。

パラメーター
なし

戻り値 所有するエンティティー名

例外 なし

MQePrivateRegistry resetPIN

構文

```
public void resetPIN(String currentPIN,
                     String newPIN ) throws Exception
```

説明 有効な専用レジストリー所有者がアクセス PIN を変更できるようにします。

パラメーター

currentPIN 専用レジストリー用に現在有効な PIN (パスワード (パスフレーズ))

newPIN 新しい PIN (パスワード (パスフレーズ))

戻り値 なし

例外

MQeException `Except _PrivateReg_BadPIN , ″PIN not reset, bad current PIN provided″`

MQePrivateRegistry setTargetRegistryName

構文

```
public void setTargetRegistryName( String registryName)
```

説明 意図した受信側の専用レジストリーの名前を追加します。

注: このメソッドは使用しないでください。代わりに、クラス MQeMTrustAttribute 内の **setTarget()** を使用してください。

パラメーター

registryName 受信側の専用レジストリーの名前

戻り値 なし

例外 なし

例 303ページの『MQeMTrustAttribute』を参照してください。

関連する関数

- MQeMTrustAttribute

MQePrivateRegistryConfigure

このクラスは、専用レジストリーを構成するときに使います。このクラスを使い、レジストリーの新しいクリデンシャル (専用および公開証明書) を取得できます。

パッケージ　**com.ibm.mqe.registry**

このクラスは、MQeRegistry の下位クラスです。

コンストラクターの要約

MQePrivateRegistryConfigure

構文

1. `public MQePrivateRegistryConfigure( )`
2. `public MQePrivateRegistryConfigure( String name,`
   `MQeFields parms,`
   `String PIN ) throws Exception`

説明

このコンストラクターは、レジストリー構成オブジェクトをインスタンス化します。これには 2 つのバージョンがあります。

1. これは空のコンストラクターであり、動的ロード用に設計されています。動的ロードの後に **activate()** を呼び出す必要があります。
2. これは名前を保管してレジストリーをオープンします。その後で **activate()** を呼び出すのは、空のコンストラクターの場合と同じです。

パラメーター

name　このレジストリーに関連付けられた名前

regParams　レジストリーの初期設定パラメーターを含む MQeFields オブジェクト

MQeRegistry.LocalRegType (ascii)
これにより、オープンするレジストリーのタイプが決まります。これは専用レジストリーにするので、このパラメーターは `com.ibm.mqe.registry.MQePrivateSession` か、同等の別名に設定する必要があります。

MQeRegistry.DirName (ascii)
レジストリー・ファイルを入れておくディレクトリーの名前

MQeRegistry.PIN (ascii)
専用レジストリーの PIN。これは、**activate()** での **regPIN** パラメーターがヌル の場合に使われます。

MQeRegistry.KeyRingPassword (ascii)
レジストリーの秘密鍵を保護するために使うパスワード (パスフレーズ)。

MQeRegistry.Separator (ascii)

特定項目名のコンポーネント間で区切り文字として使う文字 (たとえば、<QueueManager><Separator><Queue>)。

これはストリングとして指定されますが、1 つの文字だけしか含められません。複数の文字を含めると、最初の文字だけが使われます。

レジストリーをオープンするたびに同じ区切り文字を使うようにします。レジストリーの使用を開始して項目を含めたら、区切り文字を変更してはなりません。

この値を指定しないと、デフォルトの "+" が使われます。

regPIN レジストリーをオープンするときに必要な PIN。これが ヌル であれば、 PIN は **regParams** パラメーターと同じになります。

戻り値 なし

例外

例外 レジストリーのオープンに問題があれば示されます。

例

```
MQePrivateRegistryConfigure regConfig1;
regConfig1 = new MQePrivateRegistryConfigure();

try
{
 MQePrivateRegistryConfigure regConfig2;
 MQeFields parms = new MQeFields();
 parms.putAscii(MQeRegistry.DirName, "Registry_Dir");
 ...
 regConfig2 = new MQePrivateRegistryConfigure("Reg2", parms, null);
}
catch (Exception e)
{ ... }
```

メソッド

メソッド	目的
activate	クラスを初期設定し、レジストリーをオープンします。
close	レジストリーをクローズして整理します。
credentialsExist	レジストリーのクリデンシャルがすでに存在するかどうかを調べます。
getCredentials	レジストリーの新しいクリデンシャルを取得します。
isPrivate	レジストリーが専用レジストリーかどうかを調べます。
renewCertificates	レジストリーの共通証明書を更新します。

MQePrivateRegistryConfigure activate

構文

```
public void activate( String name,
                      MQeFields regParams,
                      String regPIN ) throws Exception
```

説明 レジストリー名を保管して、そのレジストリーをオープンします。 **regPIN** パラメーターが ヌル でなければ、レジストリーをオープンするために使われ、ヌル であれば、レジストリーの PIN は **regParams** パラメーターと同じになります。

パラメーター

name このレジストリーに関連付けられた名前

regParams レジストリーの初期設定パラメーターを含む MQeFields オブジェクト

MQeRegistry.LocalRegType (ascii)
これにより、オープンするレジストリーのタイプが決まります。これは専用レジストリーにするので、このパラメーターは `com.ibm.mqe.registry.MQePrivateSession` か、同等の別名に設定する必要があります。

MQeRegistry.DirName (ascii)
レジストリー・ファイルを入れておくディレクトリーの名前

MQeRegistry.PIN (ascii)
専用レジストリーの PIN。 **regPIN** パラメーターがヌル である場合に使われます。

MQeRegistry.KeyRingPassword (ascii)
レジストリーの秘密鍵を保護するために使うパスワード (パスフレーズ)。

MQeRegistry.Separator (ascii)

特定項目名のコンポーネント間で区切り文字として使う文字 (たとえば、<QueueManager><Separator><Queue>)。

これはストリングとして指定されますが、1 つの文字だけしか含められません。複数の文字を含めると、最初の文字だけが使われます。

レジストリーをオープンするたびに同じ区切り文字を使うようにします。レジストリーの使用を開始して項目を含めたら、区切り文字を変更してはなりません。

この値を指定しないと、デフォルトの "+" が使われます。

regPIN レジストリーをオープンするときに必要な PIN。これが ヌル であれば、 PIN は **regParams** パラメーターと同じになります。

戻り値 なし

例外

MQeException	レジストリーのオープンに問題があれば示されます。
例外	他に問題があれば示されます。

例

```
try
{
 MQePrivateRegistryConfigure regConfig;
 MQeFields parms = new MQeFields();
 parms.putAscii(MQeRegistry.DirName, "Registry_Dir");
 ...
 regConfig = new MQePrivateRegistryConfigure();
 regConfig.activate("Reg", parms, null);
}
catch (Exception e)
{ ... }
```

関連する関数

MQeLocalSecure

MQePrivateRegistryConfigure close

構文

```
public void close( )
```

説明 構成オブジェクトおよび関連したレジストリーをクローズします。オブジェクトのクローズ後に使おうとすると、例外が生じます。

パラメーター

なし

戻り値 なし

例外 なし

例

```
try
{
 MQePrivateRegistryConfigure regConfig;
 MQeFields parms = new MQeFields();
 parms.putAscii(MQeRegistry.DirName, "Registry_Dir");
 ...
 regConfig = new MQePrivateRegistryConfigure("Reg", parms, null);
 if ( regConfig.credentialsExist() )
 {
  ...
 }
 regConfig.close();
}
catch (Exception e)
{ ... }
```

MQePrivateRegistryConfigure credentialsExist

構文

```
public boolean credentialsExist ( ) throws MQeException
```

説明 レジストリーにクリデンシャルが含まれているかどうかを調べます。

パラメーター

なし

戻り値

true	レジストリーにクリデンシャルが含まれている場合
false	レジストリーにクリデンシャルが含まれていない場合

例外

MQeException	クラスがアクティブになっていない場合に示されます。

例

```
try
{
 MQePrivateRegistryConfigure regConfig;
 MQeFields parms = new MQeFields();
 parms.putAscii(MQeRegistry.DirName, "Registry_Dir");
 ...
 regConfig = new MQePrivateRegistryConfigure("Reg", parms, null);
 if ( regConfig.credentialsExist() )
 {
  ...
 }
}
catch (Exception e)
{ ... }
```

MQePrivateRegistryConfigure getCredentials

構文

```
public void getCredentials( MQeFields regParams,
                            String regPIN,
                            String minCertServer,
                            String miniCertPIN,
                            String renamePrefix ) throws Exception
```

説明

レジストリーの新しいクリデンシャルが作成されます。

レジストリーにクリデンシャルが含まれていれば、 **renamePrefix** で名前変更されます。名前変更に失敗すると (たとえば、新しい名前がすでにレジストリーに存在しているなど)、例外が示され、新しいクリデンシャルは取得されません。クリデンシャルの名前変更後にエラーが発生すると、 **getCredentials()** が戻される前に、元の名前に変更しなおされます。

このメソッドは最小限の証明サーバーを呼び出すので、完了するのにしばらくかかる場合があります。

パラメーター

regParams レジストリーの初期設定パラメーターを含む MQeFields オブジェクト

MQeRegistry.LocalRegType (ascii)
これにより、オープンするレジストリーのタイプが決まります。これは専用レジストリーにするので、このパラメーターは `com.ibm.mqe.registry.MQePrivateSession` か、同等の別名に設定する必要があります。

MQeRegistry.DirName (ascii)
レジストリー・ファイルを入れておくディレクトリーの名前

MQeRegistry.PIN (ascii)
専用レジストリーの PIN。これは、**activate()** での **regPIN** パラメーターがヌル の場合に使われます。

MQeRegistry.KeyRingPassword (ascii)
レジストリーの秘密鍵を保護するために使うパスワード (パスフレーズ)。

MQeRegistry.Separator (ascii)

特定項目名のコンポーネント間で区切り文字として使う文字 (たとえば、<QueueManager><Separator><Queue>)。

これはストリングとして指定されますが、1 つの文字だけしか含められません。複数の文字を含めると、最初の文字だけが使われます。

レジストリーをオープンするたびに同じ区切り文字を使うようにします。レジストリーの使用を開始して項目を含めたら、区切り文字を変更してはなりません。

この値を指定しないと、デフォルトの ″+″ が使われます。

regPIN レジストリーをオープンするときに必要な PIN。これが ヌル であれば、 PIN は **regParams** パラメーターと同じになります。

minCertServer 最小限の証明サーバーの TCP アドレスおよびポート番号

miniCertPIN レジストリーがクリデンシャルを獲得できるようにするために、最小限の証明の管理者によって事前割り振りされた*認証要求番号*

renamePrefix 既存のクリデンシャルを名前変更するときに使うプレフィックス (あれば)

戻り値 なし

例外

MQeException クラスがアクティブになっていない場合、専用レジストリーではない場合、名前変更に失敗した場合、クリデンシャルの獲得でエラーがあった場合 (たとえば、発行最小限の証明サーバーに接続したなど) に示されます。

例外 他のエラーの場合に示されます。

例

```
try
{
 MQePrivateRegistryConfigure regConfig;
 MQeFields parms = new MQeFields();
 parms.putAscii(MQeRegistry.DirName, "Registry_Dir");
 ...
 regConfig = new MQePrivateRegistryConfigure("Reg", parms, null);
 if ( regConfig.isPrivate() )
 {
   String renamePref = Long.toString(new Date().getTime()) + "_";
   regConfig.getCredentials( parms,
                             "MYpin 123",
                             "certServer.hursley.ibm.com:8082",
                             "12345678",
                             renamePref );
 }
}
catch (Exception e)
{ ... }
```

MQePrivateRegistryConfigure isPrivate

構文

```
public boolean isPrivate( ) throws MQeException
```

説明 オープンしたレジストリーが専用レジストリーであるかどうかを調べます。

パラメーター

なし

戻り値

true	専用レジストリーの場合
false	専用レジストリーではない場合

例外

MQeException	クラスがアクティブになっていない場合に示されます。

例

```
try
{
 MQePrivateRegistryConfigure regConfig;
 MQeFields parms = new MQeFields();
 parms.putAscii(MQeRegistry.DirName, "Registry_Dir");
 ...
 regConfig = new MQePrivateRegistryConfigure("Reg", parms, null);
 if ( regConfig.isPrivate() )
 {
  ...
```

```
 }
}
catch (Exception e)
{ ... }
```

MQePrivateRegistryConfigure renewCertificates

構文

```
public void renewCertificates(String regPIN,
                              String minCertServer,
                              String miniCertPIN,
                              String renamePrefix ) throws Exception
```

説明

このメソッドは、レジストリーの共通証明書を更新します。これは、たとえば、既存の証明書が期限切れになった場合に必要です。

既存の共通証明書は、*renamePrefix* を使って名前変更します。名前変更に失敗すると (たとえば、新しい名前がすでにレジストリーに存在しているなど)、例外が出され、証明書は更新されません。証明書の更新後にエラーが発生すると、 **renewCertificates()** が戻される前に、元の名前に変更しなおされます。

このメソッドは最小限の証明サーバーを呼び出すので、完了するのにしばらくかかる場合があります。

パラメーター

RegPIN レジストリーをオープンするのに必要な PIN

MinCertServer 最小限の証明サーバーの TCP アドレスおよびポート番号

MiniCertPIN レジストリーが証明書を更新できるようにするために、最小限の証明の管理者によって事前割り振りされた認証要求番号

RenamePrefix 既存の証明書を名前変更するときに使うプレフィックス

戻り値

なし

例外

MQeException クラスがアクティブになっていない場合、専用レジストリーではない場合、名前変更に失敗した場合、証明書の更新でエラーがあった場合 (たとえば、発行サーバーに接続したなど) に示されます。

例外 他のエラーの場合に示されます。

例

```
{
MQePrivateRegistryConfigure regConfig;
MQeFields parms = new MQeFields();
```

```
Parms.putAscii(MQeRegistry.DirName, "Registry_Dir");
...
regConfig = new MQePrivateRegistryConfigure("Reg", parms, null);
if (regConfig.isPrivate())
{
String renamePref = Long.toString(new Date().getTime())+"_";
regConfig.renewCertificates("MYpin 123",
"certServer.hursley.ibm.com:8082",
"12345678",
renamePref );
}
}
catch (Exception e)
{...}
```

MQePublicRegistry

このクラスは、MQePubliceRegistry オブジェクトを作成するときに使います。

パッケージ **com.ibm.mqe.registry**

このクラスは、MQeRegistry の下位クラスです。

コンストラクター

MQePublicRegistry

構文

```
public MQePublicRegistry( )
```

説明 MQePublicRegistry オブジェクトを構成します。

パラメーター

なし

戻り値 なし

例外 なし

関連する関数

- MQePrivateRegistry

メソッドの要約

メソッド	目的
activate	MQePublicRegistry インスタンスをオープンしてアクティブにします。
deleteCertificate	証明書所有者の最小限の証明を削除します。
getCertificate	証明書所有者の最小限の証明を戻します。
putCertificate	証明書所有者の最小限の証明を公開レジストリーへ追加します。
requestCertificate	別の MQSeries Everyplace の公開レジストリーから最小限の証明を要求します。
shareCertificate	証明書所有者の最小限の証明を別の MQSeries Everyplace ノードの公開レジストリーへ複製します。

MQePublicRegistry activate

構文

```
public void activate (String name,
                      String dirName) throws Exception
```

説明 このエンティティー名の公開レジストリーがある場合、 **activate** は既存の公開レジストリーをオープンし、ない場合は、 **name** という名前の新しい公開レジストリーが作られます。

パラメーター

name 公開レジストリー名 (通常は MQeNode_PublicRegistry)

dirName 公開レジストリーへのパス

戻り値 なし

例外

MQeException Except_Public_ActivateFailed, ″exception reason″

例

```
class MySampleClass extends MQe
{
try
  {
  /* setup Public Registry activate parameters          */
  String name                = "MQeNode_PublicRegistry";
  String dirName             = ".//"
  /* instantiate and activate Public Registry                      */
  MQePublicRegistry pubreg   = new MQePublicRegistry( );
  pubreg.activate( name, dirName );
  }
catch ( Exception e )
  {
  ...
  }
```

関連する関数

MQePrivateRegistry

MQePublicRegistry deleteCertificate

構文

```
public void deleteCertificate( String certificateOwner )
                                                  throws MQeException
```

説明 証明書所有者の最小限の証明を削除します。

パラメーター

certificateOwner

Mini-certificate の所有者の名前

戻り値 なし

例外

MQeException	Except_Reg_DoesNotExist, ″Entry does not exist″
	Except_Reg_DeleteFailed, ″Error deleting entry″

関連する関数

- `getCertificate`
- `putCertificate`

MQePublicRegistry getCertificate

構文

```
public MQeFields getCertificate( String certificateOwner )
                                                    throws MQeException
```

説明 証明書所有者の最小限の証明を戻します。

パラメーター

certificateOwner
認証可能なエンティティーの (最小限の証明所有者の) 名前

戻り値 Mini-certificate

例外

MQeException	Except_Reg_ReadFailed, ″Error reading entry″

関連する関数

- `deleteCertificate`
- `putCertificate`

MQePublicRegistry putCertificate

構文

```
public void putCertificate( String certificateOwner,
                            MQeFields certificate ) throws MQeException
```

説明 証明書所有者の最小限の証明を公開レジストリーへ追加します。

パラメーター

certificateOwner
認証可能なエンティティーの (最小限の証明所有者の) 名前

certificate 所有者の最小限の証明

戻り値 なし

例外

MQeException Except_Reg_AlreadyExists, ″Entry already exists″

Except_Reg_AddFailed, ″Error adding entry″

関連する関数

- `getCertificate`
- `deleteCertificate`

MQePublicRegistry requestCertificate

構文

```
public MQeFields requestCertificate( String certificateOwner,
                                     String mqeNodeAddrPort)
                                              throws MQeException
```

説明 別の MQSeries Everyplace ノードの公開レジストリーから最小限の証明を要求し、戻されたら、この公開レジストリーに保管します。

パラメーター

certificateOwner
Mini-certificate の所有者の名前

mqeNodeAddrPort
ホーム・サーバー または代替 MQSeries Everyplace ノードの TCP アドレスおよびポート

戻り値 Mini-certificate

例外

MQeException

```
Except_Reg_DoesNotExist, "Entry does not exist"
Except_Reg_ReadFailed, "Error reading entry"
Except_Reg_AddFailed, "Error adding entry"
```

例

```
class MySampleClass extends MQe
  {
  try
    {
    /* setup RequestCertificate parameters                          */
    String homeServerAddrPort  = "homeServer.hursley.ibm.com:8082");
    entityName                 = "Bruce";
    /* instantiate and activate Public Registry                     */
    MQePublicRegistry pubreg   = new MQePublicRegistry( );
    pubreg.activate("MQeNode_PublicRegistry", ".¥¥" );
    /* request Bruce's MiniCert from Public Reg on another MQeNode  */
    MQeFields minicertf        = pubreg.getCertificate( entityName,
                                                        homeServerAddrPort);
    pubreg.close();
```

```
  }
catch ( Exception e )
  {
  ...
  }
```

関連する関数

shareCertificate

MQePublicRegistry shareCertificate

構文

```
public void shareCertificate( String certificateOwner,
                              MQeFields certificate,
                              String mqeNodeAddrPort) throws MQeException
```

説明 証明書所有者の最小限の証明を別の MQSeries Everyplace ノードの公開レジストリーに複製します。

パラメーター

certificateOwner
Mini-certificate の所有者の名前

certificate Mini-certificate

mqeNodeAddrPort
ホーム・サーバー または代替 MQSeries Everyplace ノードの TCP アドレスおよびポート

戻り値 なし

例外

MQeException	Except_Reg_DoesNotExist, ″Entry does not exist″
	Except_Reg_ReadFailed, ″Error reading entry″
	Except_Reg_AddFailed, ″Error adding entry″

例

```
{
try
  {
  /* instantiate & activate a Private Reg for Auth Entity Bruce   */
  entityName                = "Bruce;
  caIPAddrPort              = "aname.hursley.ibm.com:8082";
  MQePrivateRegistry preg   = new MQePrivateRegistry( );
  preg.activate( entityName, ".¥¥MQeNode_PrivateRegistry",
            "12345678", "It_is_a_secret", "12345678", caIPAddrPort);
  /* instantiate and activate Public Reg & save Bruce's MiniCert  */
  MQePublicRegistry pubreg  = new MQePublicRegistry( );
```

```
                pubreg.activate("MQeNode_PublicRegistry",
                                                ".¥¥MQeNode_PublicRegistry" );
                pubreg.putCertificate( entityName,
                                            preg.getCertificate( entityName ) );
                /* share Bruce's MiniCert with Public Reg on another MQeNode    */
                String homeServerAddrPort  = "homeServer.hursley.ibm.com:8082");
                pubreg.shareCertificate( entityName,
                          preg.getCertificate( entityName ), homeServerAddrPort);
                preg.close();
                pubreg.close();
                }
              catch ( Exception e )
                {
                }
```

関連する関数

requestCertificate

第6章 com.ibm.mqe.server のクラス

この節には、以下の MQSeries Everyplace クラスについての詳細が載せられています。

表 16. パッケージ com.ibm.mqe.server のクラス

クラス名	目的
MQeMiniCertIssuanceInterface	MQeMiniCertificateServerGUI のインスタンスが新しい最小限の証明の発行を管理する方法を定義するときに使います。

MQeMiniCertIssuanceInterface

このインターフェースを実装した機能は、 MQeMiniCertificateServerGUI のインスタンスが新しい最小限の証明の発行を管理する方法を定義するときに使います。デフォルトの実装では、 MQeMiniCertIssuanceManager は、最小限の証明を要求できる有効な認証可能エンティティー群の定義のリポジトリーとして、 MQeMiniCertificateRegistry を使います。この場合、MQSeries Everyplace ソリューションでは、このデータのために別のリポジトリー (たとえば、別のレジストリーやデータベース・サービス) を使うことができるものと見なされます。

パッケージ　　com.ibm.mqe.server

定数

このクラスは以下に示す定数を提供します。

```
public  final static int      IssMgr_OK
public  final static int      IssMgr_Error
```

メソッド

メソッド	目的
addAuthenticatableEntity	有効な MQSeries Everyplace ソリューションの認証可能エンティティーの名前と一度だけ使用する認証要求 PIN を追加します。
addEntityRegisteredAddress	有効な MQSeries Everyplace ソリューションの認証可能エンティティーの登録済みアドレスを追加します。
authoriseMiniCertRequest	新しい最小限の証明要求を許可します。
deleteAuthenticatableEntity	有効な MQSeries Everyplace ソリューションの認証可能エンティティーの名前と一度だけ使用する認証要求 PIN を削除します。
deleteEntityRegisteredAddress	有効な MQSeries Everyplace ソリューションの認証可能エンティティーの登録済みアドレスを削除します。
readAuthenticatableEntity	有効な MQSeries Everyplace ソリューションの認証可能エンティティーの名前と一度だけ使用する認証要求 PIN を読み取ります。
readEntityRegisteredAddress	有効な MQSeries Everyplace ソリューションの認証可能エンティティーの登録済みアドレスを読み取ります。
setRegistry	MQeRegistry リポジトリーへの参照を設定します。
updateAuthenticatableEntity	有効な MQSeries Everyplace ソリューションの認証可能エンティティーの名前と一度だけ使用する認証要求 PIN を更新します。
updateEntityRegisteredAddress	有効な MQSeries Everyplace ソリューションの認証可能エンティティーの登録済みアドレスを更新します。

MQeMiniCertIssuanceInterface addAuthenticatableEntity

構文

```
public int addAuthenticatableEntity (String entityName,String certReqPIN )
```

説明 有効な MQSeries Everyplace ソリューションの認証可能エンティティーの名前と一度だけ使用する認証要求 PIN を追加します。

パラメーター

entityName 認証可能エンティティーの名前を示すときに使うストリング

certReqPIN 認証可能エンティティーの一度だけ使う認証要求 PIN を示すときに使うストリング。

戻り値

- IssMgr_OK (正常に実行されたことを示します)
- IssMgr_Error (失敗したことを示します)

例外 なし

MQeMiniCertIssuanceInterface addEntityRegisteredAddress

構文

```
public int addEntityRegisteredAddress (String entityName,
                                       MQeFields entityRegAddr )
```

説明 新しい認証可能エンティティーの登録済みアドレスを追加します。

パラメーター

entityName 認証可能エンティティーの名前を示すときに使うストリング

entityRegAddr 認証可能エンティティーの登録済みアドレスを含む MQeFields オブジェクト。

戻り値

- IssMgr_OK (正常に実行されたことを示します)
- IssMgr_Error (失敗したことを示します)

例外 なし

MQeMiniCertIssuanceInterface authoriseMiniCertRequest

構文

```
public int authoriseMiniCertRequest (String entityName,String certReqPIN )
```

説明 新しい最小限の証明要求を許可します。

パラメーター

entityName 認証可能エンティティーの名前を示すときに使うストリング

certReqPIN 認証可能エンティティーの一度だけ使う認証要求 PIN を示すときに使うストリング。

戻り値

- IssMgr_OK (正常に実行されたことを示します)
- IssMgr_Error (失敗したことを示します)

例外 なし

MQeMiniCertIssuanceInterface deleteAuthenticatableEntity

構文

```
public int deleteAuthenticatableEntity (String entityName)
```

説明 有効な MQSeries Everyplace ソリューションの認証可能エンティティーの名前と一度だけ使用する認証要求 PIN を削除します。

パラメーター

entityName 認証可能エンティティーの名前を示すときに使うストリング

戻り値

- IssMgr_OK (正常に実行されたことを示します)
- IssMgr_Error (失敗したことを示します)

例外 なし

MQeMiniCertIssuanceInterface deleteEntityRegisteredAddress

構文

```
public int deleteEntityRegisteredAddress (String entityName)
```

説明 有効な MQSeries Everyplace ソリューションの認証可能エンティティーの登録済みアドレスを削除します。

パラメーター

entityName 認証可能エンティティーの名前を示すときに使うストリング

戻り値

- IssMgr_OK (正常に実行されたことを示します)
- IssMgr_Error (失敗したことを示します)

例外 なし

MQeMiniCertIssuanceInterface readAuthenticatableEntity

構文

```
public int authoriseMiniCertRequest (String entityName,String certReqPIN )
```

説明 有効な MQSeries Everyplace ソリューションの認証可能エンティティーの名前と一度だけ使用する認証要求 PIN を読み取ります。

パラメーター

entityName 認証可能エンティティーの名前を示すときに使うストリング

byte< > 認証可能エンティティーの一度だけ使う認証要求 PIN を示すときに使うストリング。

戻り値 認証可能エンティティーの ID を含むバイト配列またはヌル

例外 なし

MQeMiniCertIssuanceInterface readEntity RegisteredAddress

構文

```
public MQeFields readEntityRegisteredAddress (String entityName)
```

説明 有効な MQSeries Everyplace ソリューションの認証可能エンティティーの登録済みアドレスを読み取ります。

パラメーター

entityName 認証可能エンティティーの名前を示すときに使うストリング

戻り値 認証可能エンティティーの登録済みアドレスを含む MQeFields オブジェクトまたはヌル

例外 なし

MQeMiniCertIssuanceInterface SetRegistry

構文

```
public void setRegistry (MQeRegistry registry)
```

説明 MQeRegistry リポジトリー・インスタンスへの参照を設定します。

パラメーター

registry MQeRegistry インスタンス

戻り値 なし

例外 なし

MQeMiniCertIssuanceInterface updateAuthenticatableEntity

構文

```
public int updateAuthenticatableEntity (String entityName,
                                        String certReqPIN )
```

説明 有効な MQSeries Everyplace ソリューションの認証可能エンティティーの名前と一度だけ使用する認証要求 PIN を更新します。

パラメーター

entityName 認証可能エンティティーの名前を示すときに使うストリング

certReqPIN 認証可能エンティティーの一度だけ使う認証要求 PIN を示すときに使うストリング。

戻り値

- `IssMgr_OK` (正常に実行されたことを示します)
- `IssMgr_Error` (失敗したことを示します)

例外 なし

MQeMiniCertIssuanceInterface updateEntityRegisteredAddress

構文

```
public int updateEntityRegisteredAddress (String entityName,
                                          MQeFields entityRegAddr )
```

説明 新しい認証可能エンティティーの登録済みアドレスを更新します。

パラメーター

entityName 認証可能エンティティーの名前を示すときに使うストリング

entityRegAddr 認証可能エンティティーの登録済みアドレスを含む MQeFields オブジェクト。

戻り値

- `IssMgr_OK` (正常に実行されたことを示します)
- `IssMgr_Error` (失敗したことを示します)

例外 なし

第7章 com.ibm.mqe.mqemqmessage のクラス

この節には、以下の MQSeries Everyplace クラスについての詳細が載せられています。

表 17. パッケージ com.ibm.mqe.mqemqmessage のクラス

クラス名	目的
MQeMQMsgObject	MQSeries Everyplace 内で MQSeries スタイル・メッセージ・オブジェクトを表すときに使います。

MQeMQMsgObject クラス

ここでは、 MQSeries Everyplace 内の MQSeries スタイル・メッセージ・オブジェクトを表すための Java クラスについて説明します。このクラスを使用して、 MQSeries スタイル・メッセージ・オブジェクトを作成したり、読み取ったりすることができます。

このクラスには、すべての MQSeries メッセージ・ヘッダー・フィールド用の **getxxx()** メソッドと **setxxx()** メソッドがあります。ただし実際には、効率を上げるために、デフォルト以外の値に設定されているフィールドだけが、メッセージ・オブジェクトに含まれるようになっています。

パッケージ **com.ibm.mqe.mqemqmessage**

このクラスは、 MQeMsgObject の下位クラスです。

コンストラクター

MQeMQMsgObject

構文

```
public MQeMQMsgObject( ) throws Exception
```

説明 新しい MQeMQMsgObject を作成します。

パラメーター

なし

戻り値 なし

例外

java.lang.Exception	スーパークラス・コンストラクター MQeMsgObject から伝搬

例

```
  ...
  MQeMQMsgObject MQMsg = new MQeMQMsgObject();
  ...
}
catch (Exception e)
{
  ...
}
```

メソッド

メソッド	目的
dumpAllToString	メッセージからすべてのフィールド値をストリングにダンプします。

メソッド	目的
dumpToString	メッセージ・オブジェクト内のフィールド値をストリングにダンプします。
equals	バイト配列を比較して、等しいかどうかをチェックします。
getAccountingToken	メッセージ・ヘッダーからアカウンティング・トークンを取得します。
getApplicationIdData	メッセージ・ヘッダーからアプリケーション ID データを取得します。
getApplicationOriginData	メッセージ・ヘッダーからアプリケーション起点データを取得します。
getBackoutCount	メッセージ・ヘッダーからバックアウト・カウントを取得します。
getCharacterSet	メッセージ・ヘッダーからコード化した文字セット ID を取得します。
getCorrelationId	メッセージ・ヘッダーから相関 ID を取得します。
getdata	メッセージ・データを取得します。
getEncoding	メッセージ・ヘッダーからエンコードの値を取得します。
getExpiry	メッセージ・ヘッダーから期限切れの値を取得します。
getFeedback	メッセージ・ヘッダーからフィードバックの値を取得します。
getFormat	メッセージ・ヘッダーからフォーマットの値を取得します。
getGroupId	メッセージ・ヘッダーからグループ ID の値を取得します。
getMessageFlags	メッセージ・ヘッダーからメッセージ・フラグの値を取得します。
getMessageId	メッセージ・ヘッダーからメッセージ ID を取得します。
getMessageSequenceNumber	メッセージ・ヘッダーからメッセージ・シーケンス番号を取得します。
getMessageType	メッセージ・ヘッダーからメッセージ・タイプを取得します。
getOffset	メッセージ・ヘッダーからオフセットの値を取得します。
getOriginalLength	メッセージ・ヘッダーからオリジナルの長さを取得します。
getPersistence	メッセージ・ヘッダーから持続性の値を取得します。
getPriority	メッセージ・ヘッダーから優先順位を取得します。
getPutApplicationName	メッセージ・ヘッダーからアプリケーション名書き込みを取得します。
getPutApplicationType	メッセージ・ヘッダーからアプリケーション・タイプ書き込みを取得します。

メソッド	目的
getPutDateTime	メッセージ・ヘッダーから日時書き込みを取得します。
getReplyToQueueManagerName	メッセージ・ヘッダーから応答先キュー・マネージャー名を取得します。
getReplyToQueueName	メッセージ・ヘッダーから応答先キュー名を取得します。
getReport	メッセージ・ヘッダーからレポートの値を取得します。
getUserId	メッセージ・ヘッダーからユーザー ID を取得します。
setAccountingToken	メッセージ・ヘッダーにアカウンティング・トークンの値を設定します。
setApplicationIdData	メッセージ・ヘッダーにアプリケーション ID データを設定します。
setApplicationOriginData	メッセージ・ヘッダーにアプリケーション起点データを設定します。
setBackoutCount	メッセージ・ヘッダーにバックアウト・カウントを設定します。
setCharacterSet	メッセージ・ヘッダーにコード化した文字セット ID を設定します。
setCorrelationId	メッセージ・ヘッダーに相関 ID を設定します。
setdata	メッセージ・データを設定します。
setEncoding	メッセージ・ヘッダーにエンコードの値を設定します。
setExpiry	メッセージ・ヘッダーに期限切れの値を設定します。
setFeedback	メッセージ・ヘッダーにフィードバックの値を設定します。
setFormat	メッセージ・ヘッダーにフォーマットの値を設定します。
setGroupId	メッセージ・ヘッダーにグループ ID の値を設定します。
setMessageFlags	メッセージ・ヘッダーにメッセージ・フラグの値を設定します。
setMessageId	メッセージ・ヘッダーにメッセージ ID を設定します。
setMessageSequenceNumber	メッセージ・ヘッダーにメッセージ・シーケンス番号を設定します。
setMessageType	メッセージ・ヘッダーにメッセージ・タイプを設定します。
setOffset	メッセージ・ヘッダーにオフセットの値を設定します。
setOriginalLength	メッセージ・ヘッダーにオリジナルの長さを設定します。
setPersistence	メッセージ・ヘッダーに持続性の値を設定します。
setPriority	メッセージ・ヘッダーに優先順位を設定します。
setPutApplicationName	メッセージ・ヘッダーにアプリケーション名書き込みを設定します。
setPutApplicationType	メッセージ・ヘッダーにアプリケーション・タイプ書き込みを設定します。

メソッド	目的
setPutDateTime	メッセージ・ヘッダーに日時書き込みを設定します。
setReplyToQueueManagerName	メッセージ・ヘッダーに応答先キュー・マネージャー名を設定します。
setReplyToQueueName	メッセージ・ヘッダーに応答先キュー名を設定します。
setReport	メッセージ・ヘッダーにレポートの値を設定します。
setUserId	メッセージ・ヘッダーにユーザー ID を設定します。

MQeMQMsgObject dumpAllToString

構文

```
public String dumpAllToString()
```

説明 このメソッドは、MQSeries スタイル・メッセージから、すべてのヘッダー・フィールドと、それぞれのフィールド値と、データ・フィールドの値をストリングにダンプします。デバッグ用に便利です。

このメソッドは、すべてのヘッダー・フィールドをストリングにダンプするのに対し、 **dumpToString()** メソッドは、デフォルト以外の値に設定されているフィールドだけをダンプします。

パラメーター
なし

戻り値 フィールドの名前と値、それにデータ値を含んだストリング。

例外 なし

例

```
if (msgObj instanceof MQeMQMsgObject)
{
  System.out.println(((MQeMQMsgObject)msgObj).dumpAllToString());
}
```

MQeMQMsgObject dumpToString

構文

```
public String dumpToString()
```

説明 このメソッドは、MQSeries スタイル・メッセージから、ヘッダー・フィールドと、それぞれの値と、データ・フィールドの値をストリングにダンプします。デバッグ用に便利です。

このメソッドは、デフォルト以外の値に設定されているフィールドだけをダンプするのに対し、 **dumpAllToString()** メソッドは、すべてのヘッダー・フィールドをストリングにダンプします。

パラメーター

なし

戻り値 フィールドの名前と値、それにデータ値を含んだストリング。

例外 なし

例

```
if (msgObj instanceof MQeMQMsgObject)
{
  System.out.println(((MQeMQMsgObject)msgObj).dumpToString());
}
```

MQeMQMsgObject equals

構文

```
public boolean equals(byte [] b1, byte [] b2)
```

説明 2 つのバイト配列を比較して、等しいかどうかをチェックします。両者が同じ長さで、片方の配列の各バイトが他方の配列の対応バイトと等しい場合に、その両者は等しいと見なされます。

パラメーター

b1 比較対象の最初のバイト配列

b2 比較対象の 2 番目のバイト配列

戻り値 両方のバイト配列の長さと内容が等しい場合は `true`、そうでない場合は `false`。

例外 なし

例

```
byte [] correlId = ...
if ( mqMsgObj.equals(mqMsgObj.getCorrelationId(), correlId) )
{
  ...
}
```

MQeMQMsgObject getAccountingToken

構文

```
public byte [] getAccountingToken() throws Exception
```

説明 このメソッドは、*AccountingToken* ヘッダー・フィールドの値を戻します。

パラメーター

なし

戻り値 アカウンティング・トークンの値を含んだバイト配列。

例外

java.lang.Exception	メッセージ・オブジェクトから値を読み取るときにエラーが発生した場合。

例

```
try
{
  if (msgObj instanceof MQeMQMsgObject)
  {
    MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
    byte [] accountToken = mqMsgObj.getAccountingToken();
    ...
  }
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject getApplicationIdData

構文

```
MQeMQMsgObject getApplicationIdData
```

説明 このメソッドは、*ApplIdentityData* ヘッダー・フィールドの値を戻します。

パラメーター

なし

戻り値 アプリケーション ID データの値を含んだストリング。

例外

java.lang.Exception	メッセージ・オブジェクトから値を読み取るときにエラーが発生した場合。

例

```
try
{
  if (msgObj instanceof MQeMQMsgObject)
  {
    MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
    String appIdData = mqMsgObj.getApplicationIdData();
    α
  }
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject getApplicationOriginData

構文

```
public String getApplicationOriginData() throws Exception
```

説明 このメソッドは、*ApplOriginData* ヘッダー・フィールドの値を戻します。

パラメーター

なし

戻り値 アプリケーション起点データの値を含んだストリング。

例外

java.lang.Exception	メッセージ・オブジェクトから値を読み取るときにエラーが発生した場合。

例

```
try
{
  if (msgObj instanceof MQeMQMsgObject)
  {
    MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
    String appOriginData = mqMsgObj.getApplicationOriginData();
    ...
  }
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject getBackoutCount

構文

```
public int getBackoutCount() throws Exception
```

説明 このメソッドは、*BackoutCount* ヘッダー・フィールドの値を戻します。

パラメーター

なし

戻り値 バックアウト・カウントの値を含んだ整数。

例外

java.lang.Exception	メッセージ・オブジェクトから値を読み取るときにエラーが発生した場合。

例

```
try
{
  if (msgObj instanceof MQeMQMsgObject)
  {
```

```
    MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
    int backoutCount = mqMsgObj.getBackoutCount();
    ...
  }
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject getCharacterSet

構文

```
public int getCharacterSet() throws Exception
```

説明 このメソッドは、*CodedCharSetId* ヘッダー・フィールドの値を戻します。

パラメーター

なし

戻り値 コード化した文字セット ID の値を含んだ整数。

例外

java.lang.Exception	メッセージ・オブジェクトから値を読み取るときにエラーが発生した場合。

例

```
try
{
  if (msgObj instanceof MQeMQMsgObject)
  {
    MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
    int charSet = mqMsgObj.getCharacterSet();
    ...
  }
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject getData

構文

```
public byte [] getData() throws Exception
```

説明 このメソッドは、メッセージ・データを戻します。アプリケーションは、そのデータの解釈方法を知っている必要があります。

パラメーター

なし

戻り値 メッセージ・データを含んだバイト配列。

例外

java.lang.Exception	メッセージ・オブジェクトから値を読み取るときにエラーが発生した場合。

例

```
try
{
  if (msgObj instanceof MQeMQMsgObject)
  {
    MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
    byte [] msgData = mqMsgObj.getData();
    ...
  }
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject getCorrelationId

構文

```
public byte [] getCorrelationId() throws Exception
```

説明 このメソッドは、*CorrelId* ヘッダー・フィールドの値を戻します。

パラメーター

なし

戻り値 相関 ID の値を含んだバイト配列。

例外

java.lang.Exception	メッセージ・オブジェクトから値を読み取るときにエラーが発生した場合。

例

```
try
{
  if (msgObj instanceof MQeMQMsgObject)
  {
    MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
    byte [[ correlId = mqMsgObj.getCorrelationId();
    ...
  }
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject getEncoding

構文

```
MQeMQMsgObject getEncoding
```

説明 このメソッドは、*Encoding* ヘッダー・フィールドの値を戻します。

パラメーター

なし

戻り値 エンコードの値を含んだ整数。

例外

java.lang.Exception	メッセージ・オブジェクトから値を読み取るときにエラーが発生した場合。

例

```
try
{
  if (msgObj instanceof MQeMQMsgObject)
  {
    MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
    int encode = mqMsgObj.getEncoding();
    ...
  }
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject getExpiry

構文

```
public int getExpiry() throws Exception
```

説明 このメソッドは、*Expiry* ヘッダー・フィールドの値を戻します。この値は、MQSeries メッセージの中で使用されている単位と同じく、 10 分の 1 秒単位になります (したがって、 MQSeries Everyplace の期限切れ時間の単位であるミリ秒ではありません)。

パラメーター

なし

戻り値 期限切れの値を含んだ整数。

例外

java.lang.Exception	メッセージ・オブジェクトから値を読み取るときにエラーが発生した場合。

例

```
try
{
  if (msgObj instanceof MQeMQMsgObject)
  {
    MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
    int expiry = mqMsgObj.getExpiry();
    ...
  }
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject getFeedback

構文

```
MQeMQMsgObject getFeedback
```

説明 このメソッドは、*Feedback* ヘッダー・フィールドの値を戻します。

パラメーター

なし

戻り値 フィードバックの値を含んだ整数。

例外

java.lang.Exception	メッセージ・オブジェクトから値を読み取るときにエラーが発生した場合。

例

```
try
{
  if (msgObj instanceof MQeMQMsgObject)
  {
    MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
    int feedback = mqMsgObj.getFeedback();
    ...
  }
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject getFormat

構文

```
public String getFormat() throws Exception
```

説明 このメソッドは、*Format* ヘッダー・フィールドの値を戻します。

パラメーター

なし

戻り値 フォーマットの値を含んだストリング。

例外

java.lang.Exception	メッセージ・オブジェクトから値を読み取るときにエラーが発生した場合。

例

```
try
{
  if (msgObj instanceof MQeMQMsgObject)
  {
    MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
    String format = mqMsgObj.getFormat();
    ...
  }
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject getGroupId

構文

```
public byte [] getGroupId() throws Exception
```

説明 このメソッドは、*GroupId* ヘッダー・フィールドの値を戻します。

パラメーター

なし

戻り値 グループ ID の値を含んだバイト配列。

例外

java.lang.Exception	メッセージ・オブジェクトから値を読み取るときにエラーが発生した場合。

例

```
try
{
  if (msgObj instanceof MQeMQMsgObject)
  {
    MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
    byte [] groupId = mqMsgObj.getGroupId();
    ...
  }
}
```

```
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject getMessageFlags

構文

```
public int getMessageFlags() throws Exception
```

説明 このメソッドは、*MsgFlags* ヘッダー・フィールドの値を戻します。

パラメーター

なし

戻り値 このメソッドは、メッセージ・フラグ・ヘッダー・フィールドの値を戻します。

例外

java.lang.Exception	メッセージ・オブジェクトから値を読み取るときにエラーが発生した場合。

例

```
try
{
  if (msgObj instanceof MQeMQMsgObject)
  {
    MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
    int msgFlags = mqMsgObj.getMessageFlags();
    ...
  }
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject getMessageId

構文

```
public byte [] getMessageId() throws Exception
```

説明 このメソッドは、*MsgID* ヘッダー・フィールドの値を戻します。

パラメーター

なし

戻り値 メッセージ ID の値を含むバイト配列。

例外

java.lang.Exception	メッセージ・オブジェクトから値を読み取るときにエラーが発生した場合。

例

```
try
{
  if (msgObj instanceof MQeMQMsgObject)
  {
    MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
    byte [] msgId = mqMsgObj.getMessageId();
    ...
  }
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject getMessageSequenceNumber

構文

```
public int getMessageSequenceNumber() throws Exception
```

説明 このメソッドは、*MsgSeqNumber* ヘッダー・フィールドの値を戻します。

パラメーター

なし

戻り値 メッセージ・シーケンス番号の値を含んだ整数。

例外

java.lang.Exception	メッセージ・オブジェクトから値を読み取るときにエラーが発生した場合。

例

```
try
{
  if (msgObj instanceof MQeMQMsgObject)
  {
    MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
    int msgSeqNo = mqMsgObj.getMessageSequenceNumber();
    ...
  }
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject getMessageType

構文

```
public int getMessageType() throws Exception
```

説明 このメソッドは、*MsgType* ヘッダー・フィールドの値を戻します。

パラメーター

なし

戻り値 メッセージ・タイプの値を含んだ整数。

例外

java.lang.Exception	メッセージ・オブジェクトから値を読み取るときにエラーが発生した場合。

例

```
try
{
  if (msgObj instanceof MQeMQMsgObject)
  {
    MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
    int msgType = mqMsgObj.getMessageType();
    ...
  }
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject getOffset

構文

```
public int getOffset() throws Exception
```

説明 このメソッドは、*Offset* ヘッダー・フィールドの値を戻します。

パラメーター

なし

戻り値 オフセットの値を含んだ整数。

例外

java.lang.Exception	メッセージ・オブジェクトから値を読み取るときにエラーが発生した場合。

例

```
try
{
  if (msgObj instanceof MQeMQMsgObject)
  {
```

```
    MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
    int offset = mqMsgObj.getOffset();
    ...
  }
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject getOriginalLength

構文

```
public int getOriginalLength() throws Exception
```

説明 このメソッドは、*OriginalLength* ヘッダー・フィールドの値を戻します。

パラメーター

なし

戻り値 オリジナルの長さの値を含んだ整数。

例外

java.lang.Exception	メッセージ・オブジェクトから値を読み取るときにエラーが発生した場合。

例

```
try
{
  if (msgObj instanceof MQeMQMsgObject)
  {
    MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
    int originalLength = mqMsgObj.getOriginalLength();
    ...
  }
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject getPersistence

構文

```
public int getPersistence() throws Exception
```

説明 このメソッドは、*Persistence* ヘッダー・フィールドの値を戻します。

パラメーター

なし

戻り値 持続性の値を含んだ整数。

例外

java.lang.Exception	メッセージ・オブジェクトから値を読み取るときにエラーが発生した場合。

例

```
try
{
  if (msgObj instanceof MQeMQMsgObject)
  {
    MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
    int persistence = mqMsgObj.getPersistence();
    ...
  }
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject getPriority

構文

```
public int getPriority() throws Exception
```

説明 このメソッドは、*Priority* ヘッダー・フィールドの値を戻します。

パラメーター

なし

戻り値 優先順位の値を含んだ整数。

例外

java.lang.Exception	メッセージ・オブジェクトから値を読み取るときにエラーが発生した場合。

例

```
try
{
  if (msgObj instanceof MQeMQMsgObject)
  {
    MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
    int priority = mqMsgObj.getPriority();
    ...
  }
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject getPutApplicationName

構文

```
public String getPutApplicationName() throws Exception
```

説明 このメソッドは、*PutApplName* ヘッダー・フィールドの値を戻します。

パラメーター

なし

戻り値 アプリケーション名書き込みの値を含んだストリング。

例外

java.lang.Exception	メッセージ・オブジェクトから値を読み取るときにエラーが発生した場合。

例

```
try
{
  if (msgObj instanceof MQeMQMsgObject)
  {
    MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
    String putApplName = mqMsgObj.getPutApplicationName();
    ...
  }
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject getPutApplicationType

構文

```
public int getPutApplicationType() throws Exception
```

説明 このメソッドは、*PutApplType* ヘッダー・フィールドの値を戻します。

パラメーター

なし

戻り値 アプリケーション・タイプ書き込みの値を含んだ整数。

例外

java.lang.Exception	メッセージ・オブジェクトから値を読み取るときにエラーが発生した場合。

例

```
try
{
  if (msgObj instanceof MQeMQMsgObject)
  {
```

```
    MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
    int putApplType = mqMsgObj.getPutApplicationType();
    ...
  }
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject getPutDateTime

構文

```
public GregorianCalendar getPutDateTime() throws Exception
```

説明 このメソッドは、*PutDate* および *PutTime* ヘッダー・フィールドの値を戻します。この値は、MQSeries クラス Java との整合性を得るために、グレゴリオ暦オブジェクトとして戻されます。

パラメーター

なし

戻り値 日時書き込みの値を含んだグレゴリオ暦オブジェクト。

例外

java.lang.Exception	メッセージ・オブジェクトから値を読み取るときにエラーが発生した場合。

例

```
try
{
  if (msgObj instanceof MQeMQMsgObject)
  {
    MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
    GregorianCalendar putDateTime = mqMsgObj.getPutDateTime();
    ...
  }
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject getReplyToQueueManagerName

構文

```
MQeMQMsgObject getReplyToQueueManagerName
```

説明 このメソッドは、*ReplyToQMgr* ヘッダー・フィールドの値を戻します。

パラメーター

なし

戻り値 応答先キュー・マネージャー名の値を含んだストリング。

例外

java.lang.Exception	メッセージ・オブジェクトから値を読み取るときにエラーが発生した場合。

例

```
try
{
  if (msgObj instanceof MQeMQMsgObject)
  {
    MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
    String replyToQueueMgrName = mqMsgObj.getReplyToQueueManagerName();
    ...
  }
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject getReplyToQueueName

構文

```
public String getReplyToQueueName() throws Exception
```

説明 このメソッドは、*ReplyToQ* ヘッダー・フィールドの値を戻します。

パラメーター

なし

戻り値 応答先キュー名の値を含んだストリング。

例外

java.lang.Exception	メッセージ・オブジェクトから値を読み取るときにエラーが発生した場合。

例

```
try
{
  if (msgObj instanceof MQeMQMsgObject)
  {
    MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
    String replyToQueueName = mqMsgObj.getReplyToQueueName();
    ...
  }
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject getReport

構文

```
public int getReport() throws Exception
```

説明 このメソッドは、*Report* ヘッダー・フィールドの値を戻します。

パラメーター

なし

戻り値 レポートの値を含んだ整数。

例外

java.lang.Exception	メッセージ・オブジェクトから値を読み取るときにエラーが発生した場合。

例

```
try
{
  if (msgObj instanceof MQeMQMsgObject)
  {
    MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
    int report = mqMsgObj.getReport();
    ...
  }
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject getUserId

構文

```
public String getUserId() throws Exception
```

説明 このメソッドは、*UserIdentifier* ヘッダー・フィールドの値を戻します。

パラメーター

なし

戻り値 ユーザー ID の値を含んだストリング。

例外

java.lang.Exception	メッセージ・オブジェクトから値を読み取るときにエラーが発生した場合。

例

```
try
{
  if (msgObj instanceof MQeMQMsgObject)
  {
```

```
    MQeMQMsgObject mqMsgObj = (MQeMQMsgObject)msgObj;
    String userId = mqMsgObj.getUserId();
    ...
  }
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject setAccountingToken

構文

```
public void setAccountingToken(byte [] accountingToken) throws Exception
```

説明 このメソッドは、MQSeries スタイル・メッセージに *AccountingToken* ヘッダー・フィールドの値を設定します。

パラメーター

accountingToken
アカウンティング・トークン・フィールドに設定する値を含んだバイト配列。

戻り値 なし

例外

java.lang.Exception メッセージ・オブジェクトに値を設定するときにエラーが発生した場合。

例

```
try
{
  MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
  byte [] accountingToken = ...;
  mqeMsgObj.setAccountingToken(accountingToken);
  ...
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject setApplicationIdData

構文

```
public void setApplicationIdData(String applicationIdData) throws Exception
```

説明 このメソッドは、MQSeries スタイル・メッセージに *ApplIdData* ヘッダー・フィールドの値を設定します。

パラメーター

applicationIdData
アプリケーション ID データ・フィールドに設定する値を含んだストリング。

戻り値 なし

例外

java.lang.Exception メッセージ・オブジェクトに値を設定するときにエラーが発生した場合。

例

```
try
{
  MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
  String applIdData = ...;
  mqeMsgObj.setApplicationIdData(applIdData);
  ...
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject setApplicationOriginData

構文

```
public void setApplicationOriginData(String applicationOriginData) throws Exception
```

説明 このメソッドは、MQSeries スタイル・メッセージに *ApplOriginData* ヘッダー・フィールドの値を設定します。

パラメーター

applicationOriginData
アプリケーション起点データ・フィールドに設定する値を含んだストリング。

戻り値 なし

例外

java.lang.Exception メッセージ・オブジェクトに値を設定するときにエラーが発生した場合。

例

```
try
{
  MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
  String applOriginData = ...;
  mqeMsgObj.setApplicationOriginData(applOriginData);
  ...
}
```

```
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject setBackoutCount

構文

```
public void setBackoutCount(int backoutCount) throws Exception
```

説明 このメソッドは、MQSeries スタイル・メッセージに *BackoutCount* ヘッダー・フィールドの値を設定します。

パラメーター

backoutCount バックアウト・カウント・フィールドに設定する値を含んだ整数。

戻り値 なし

例外

java.lang.Exception メッセージ・オブジェクトに値を設定するときにエラーが発生した場合。

例

```
try
{
  MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
  int backoutCount = ...;
  mqeMsgObj.setBackoutCount(backoutCount);
  ...
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject setCharacterSet

構文

```
public void setCharacterSet(int characterSet) throws Exception
```

説明 このメソッドは、MQSeries スタイル・メッセージに *CodedCharSetId* ヘッダー・フィールドの値を設定します。

パラメーター

characterSet コード化された文字セット ID フィールドに設定する値を含んだ整数。

戻り値 なし

例外

java.lang.Exception メッセージ・オブジェクトに値を設定するときにエラーが発生した場合。

例

```
try
{
  MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
  int characterSet = ...;
  mqeMsgObj.setCharacterSet(characterSet);
  ...
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject setCorrelationId

構文

```
public void setCorrelationId(byte [] correlationId) throws Exception
```

説明 このメソッドは、MQSeries スタイル・メッセージに *CorrelId* ヘッダー・フィールドの値を設定します。さらに、MQSeries Everyplace システム自体の内部で使用する相関 ID も設定します。

パラメーター

correlationId 相関 ID フィールドに設定する値を含んだバイト配列。

戻り値 なし

例外

java.lang.Exception メッセージ・オブジェクトに値を設定するときにエラーが発生した場合。

例

```
try
{
  MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
  byte [] correlationId = ...;
  mqeMsgObj.setCorrelationId(correlationId);
  ...
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject setData

構文

```
public void setData(byte [] data) throws Exception
```

説明 このメソッドは、 MQSeries スタイル・メッセージにメッセージ・データを設定します。

パラメーター

data メッセージ・データを含んだバイト配列。

戻り値 なし

例外

java.lang.Exception メッセージ・オブジェクトに値を設定するときにエラーが発生した場合。

例

```
try
{
  MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
  byte [] data = ...;
  mqeMsgObj.setData(data);
  ...
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject setEncoding

構文

```
public void setEncoding(int encoding) throws Exception
```

説明 このメソッドは、MQSeries スタイル・メッセージに *Encoding* ヘッダー・フィールドの値を設定します。

パラメーター

encoding エンコード・フィールドに設定する値を含んだ整数。

戻り値 なし

例外

java.lang.Exception メッセージ・オブジェクトに値を設定するときにエラーが発生した場合。

例

```
try
{
  MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
  int encoding = ...;
  mqeMsgObj.setEncoding(encoding);
  ...
}
```

```
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject setExpiry

構文

```
public void setExpiry(int expiry) throws Exception
```

説明 このメソッドは、MQSeries スタイル・メッセージに *Expiry* ヘッダー・フィールドの値を設定します。さらに、 MQSeries Everyplace システム自体の内部で使用するメッセージ期限切れ時間も設定します。

パラメーター

expiry 期限切れフィールドに設定する値を含んだ整数。この値は、MQSeries メッセージの中で使用されている単位と同じく、10 分の 1 秒単位にすべきです (したがって、 MQSeries Everyplace の期限切れ時間の単位であるミリ秒ではありません)。

戻り値 なし

例外

java.lang.Exception メッセージ・オブジェクトに値を設定するときにエラーが発生した場合。

例

```
try
{
  MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
  int expiry = ...;
  mqeMsgObj.setExpiry(expiry);
  ...}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject setFeedback

構文

```
public void setFeedback(int feedback) throws Exception
```

説明 このメソッドは、MQSeries スタイル・メッセージに *Feedback* ヘッダー・フィールドの値を設定します。

パラメーター

feedback フィードバック・フィールドに設定する値を含んだ整数。

戻り値 なし

例外

java.lang.Exception	メッセージ・オブジェクトに値を設定するときにエラーが発生した場合。

例

```
try
{
  MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
  int feedback = ...;
  mqeMsgObj.setFeedback(feedback);
  ...
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject setFormat

構文

```
public void setFormat(String format) throws Exception
```

説明 このメソッドは、MQSeries スタイル・メッセージに *Format* ヘッダー・フィールドの値を設定します。

パラメーター

format	フォーマット・フィールドに設定する値を含んだストリング。

戻り値 なし

例外

java.lang.Exception	メッセージ・オブジェクトに値を設定するときにエラーが発生した場合。

例

```
try
{
  MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
  String format = ...;
  mqeMsgObj.setFormat(format);
  ...
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject setGroupId

構文

```
public void setGroupId(byte [] groupId) throws Exception
```

説明 このメソッドは、MQSeries スタイル・メッセージに *GroupId* ヘッダー・フィールドの値を設定します。

パラメーター

groupId グループ ID フィールドに設定する値を含んだバイト配列。

戻り値 なし

例外

java.lang.Exception メッセージ・オブジェクトに値を設定するときにエラーが発生した場合。

例

```
try
{
  MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
  byte [] groupId = ...;
  mqeMsgObj.setGroupId(groupId);
  ...
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject setMessageFlags

構文

```
public void setMessageFlags(int messageFlags) throws Exception
```

説明 このメソッドは、MQSeries スタイル・メッセージに *MsgFlags* ヘッダー・フィールドの値を設定します。

パラメーター

format メッセージ・フラグ・フィールドに設定する値を含んだストリング。

戻り値 なし

例外

java.lang.Exception メッセージ・オブジェクトに値を設定するときにエラーが発生した場合。

例

```
try
{
  MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
  int messageFlags = ...;
  mqeMsgObj.setMessageFlags(messageFlags);
  ...
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject setMessageId

構文

```
public void setMessageId(byte [] messageId) throws Exception
```

説明 このメソッドは、MQSeries スタイル・メッセージに *MsgId* ヘッダー・フィールドの値を設定します。

パラメーター

messageId メッセージ ID フィールドに設定する値を含んだバイト配列。

戻り値 なし

例外

java.lang.Exception メッセージ・オブジェクトに値を設定するときにエラーが発生した場合。

例

```
try
{
  MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
  byte [] messageId = ...;
  mqeMsgObj.setMessageId(messageId);
  ...
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject setMessageSequenceNumber

構文

```
public void setMessageSequenceNumber(int seqNo) throws Exception
```

説明 このメソッドは、MQSeries スタイル・メッセージに *MsgSeqNumber* ヘッダー・フィールドの値を設定します。

パラメーター

seqNo メッセージ・シーケンス番号フィールドに設定する値を含んだ整数。

戻り値 なし

例外

java.lang.Exception メッセージ・オブジェクトに値を設定するときにエラーが発生した場合。

例

```
try
{
  MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
  int seqNo = ...;
  mqeMsgObj.setMessageSequenceNumber(seqNo);
  ...
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject setMessageType

構文

```
public void setMessageType(int messageType) throws Exception
```

説明 このメソッドは、MQSeries スタイル・メッセージに *MsgType* ヘッダー・フィールドの値を設定します。さらに、 MQSeries Everyplace システム自体の内部で使用するメッセージ・スタイルも設定します。

パラメーター

messageType メッセージ・タイプ・フィールドに設定する値を含んだ整数。

戻り値 なし

例外

java.lang.Exception メッセージ・オブジェクトに値を設定するときにエラーが発生した場合。

例

```
try
{
  MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
  int messageType = ...;
  mqeMsgObj.setMessageType(messageType);
  ...
}
```

```
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject setOffset

構文

```
public void setOffset(int offset) throws Exception
```

説明 このメソッドは、MQSeries スタイル・メッセージに *Offset* ヘッダー・フィールドの値を設定します。

パラメーター

offset オフセット・フィールドに設定する値を含んだ整数。

戻り値 なし

例外

java.lang.Exception メッセージ・オブジェクトに値を設定するときにエラーが発生した場合。

例

```
try
{
  MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
  int offset = ...;
  mqeMsgObj.setOffset(offset);
  ...
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject setOriginalLength

構文

```
public void setOriginalLength(int len) throws Exception
```

説明 このメソッドは、MQSeries スタイル・メッセージに *OriginalLength* ヘッダー・フィールドの値を設定します。

パラメーター

len オリジナルの長さフィールドに設定する値を含んだ整数。

戻り値 なし

例外

java.lang.Exception メッセージ・オブジェクトに値を設定するときにエラーが発生した場合。

例

```
try
{
  MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
  int len = ...;
  mqeMsgObj.setOriginalLength(len);
  ...
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject setPersistence

構文

```
public void setPersistence(int persistence) throws Exception
```

説明 このメソッドは、MQSeries スタイル・メッセージに *Persistence* ヘッダー・フィールドの値を設定します。

パラメーター

persistence 持続性フィールドに設定する値を含んだ整数。

戻り値 なし

例外

java.lang.Exception メッセージ・オブジェクトに値を設定するときにエラーが発生した場合。

例

```
try
{
  MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
  int persistence = ...;
  mqeMsgObj.setPersistence(persistence);
  ...
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject setPriority

構文

```
public void setPriority(int priority) throws Exception
```

説明 このメソッドは、MQSeries スタイル・メッセージに *Priority* ヘッダー・フィールドの値を設定します。さらに、 MQSeries Everyplace システム自体の内部で使用するメッセージ優先順位も設定します。

パラメーター

priority 優先順位フィールドに設定する値を含んだ整数。この値は、0 ～ 9 にしてください。

戻り値 なし

例外

java.lang.Exception メッセージ・オブジェクトに値を設定するときにエラーが発生した場合。

例

```
try
{
  MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
  int priority = ...;
  mqeMsgObj.setPriority(priority);
  ...
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject setPutApplicationName

構文

```
public void setPutApplicationName(String putApplicationName) throws Exception
```

説明 このメソッドは、MQSeries スタイル・メッセージに *PutApplName* ヘッダー・フィールドの値を設定します。

パラメーター

putApplicationName
アプリケーション名書き込みフィールドに設定する値を含んだストリング。

戻り値 なし

例外

java.lang.Exception メッセージ・オブジェクトに値を設定するときにエラーが発生した場合。

例

```
try
{
  MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
  String putApplName = ...;
  mqeMsgObj.setPutApplicationName(putApplName);
  ...
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject setPutApplicationType

構文

```
public void setPutApplicationType(int putApplicationType) throws Exception
```

説明 このメソッドは、MQSeries スタイル・メッセージに *PutApplType* ヘッダー・フィールドの値を設定します。

パラメーター

putApplicationType
アプリケーション・タイプ書き込みフィールドに設定する値を含んだ整数。

戻り値 なし

例外

java.lang.Exception	メッセージ・オブジェクトに値を設定するときにエラーが発生した場合。

例

```
try
{
  MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
  int putAppltype = ...;
  mqeMsgObj.setPutApplicationType(putApplType);
  ...
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject setPutDateTime

構文

```
public void setPutDateTime(GregorianCalendar calendar) throws Exception
```

説明 このメソッドは、 MQSeries スタイル・メッセージに *PutDate* および *PutTime*

ヘッダー・フィールドの値を設定します。 MQSeries クラス Java との整合性を得るために、日時の指定には GregorianCalendar オブジェクトを使用します。

パラメーター

calendar 日時書き込みフィールドに設定する値を含んだ GregorianCalendar オブジェクト。

戻り値 なし

例外

java.lang.Exception メッセージ・オブジェクトに値を設定するときにエラーが発生した場合。

例

```
try
{
  MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
  GregorianCalendar calendar = ...;
  mqeMsgObj.setPutDateTime(calendar);
  ...
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject setReplyToQueueManagerName

構文

```
public void setReplyToQueueManagerName(String replyToQMName) throws Exception
```

説明 このメソッドは、MQSeries スタイル・メッセージに *ReplyToQMgr* ヘッダー・フィールドの値を設定します。

パラメーター

replyToQMName
応答先キュー・マネージャー名フィールドに設定する値を含んだストリング。

戻り値 なし

例外

java.lang.Exception メッセージ・オブジェクトに値を設定するときにエラーが発生した場合。

例

```
try
{
  MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
  String replyToQMName = ...;
  mqeMsgObj.setReplyToQueueManagerName(replyToQMName);
  ...
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject setReplyToQueueName

構文

```
public void setReplyToQueueName(String replyToQueueName) throws Exception
```

説明 このメソッドは、MQSeries スタイル・メッセージに *ReplyToQ* ヘッダー・フィールドの値を設定します。

パラメーター

replyToQueueName
応答先キュー名フィールドに設定する値を含んだストリング。

戻り値 なし

例外

java.lang.Exception メッセージ・オブジェクトに値を設定するときにエラーが発生した場合。

例

```
try
{
  MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
  String replyToQueueName = [;
  mqeMsgObj.setReplyToQueueName(replyToQueueName);
  ...
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject setReport

構文

```
public void setReport(int report) throws Exception
```

説明 このメソッドは、MQSeries スタイル・メッセージに *Report* ヘッダー・フィールドの値を設定します。

パラメーター

report レポート・フィールドに設定する値を含んだ整数。

戻り値 なし

例外

java.lang.Exception メッセージ・オブジェクトに値を設定するときにエラーが発生した場合。

例

```
try
{
  MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
  int report = ...;
  mqeMsgObj.setReport(report);
  ...
}
catch (Exception e)
{
  ...
}
```

MQeMQMsgObject setUserId

構文

```
public void setUserId(String userId) throws Exception
```

説明 このメソッドは、MQSeries スタイル・メッセージに *UserIdentifier* ヘッダー・フィールドの値を設定します。

パラメーター

userId ユーザー ID フィールドに設定する値を含んだストリング。

戻り値 なし

例外

java.lang.Exception メッセージ・オブジェクトに値を設定するときにエラーが発生した場合。

例

```
try
{
  MQeMQMsgObject mqeMsgObj = new MQeMQMsgObject();
  String userId = ...;
  mqeMsgObj.setUserId(userId);
  ...
}
catch (Exception e)
{
  ...
}
```

第8章 com.ibm.mqe.mqbridge のクラス

ここでは、 MQSeries Everyplace の以下のクラスとインターフェースについて詳しく説明します。

表 18. com.ibm.mqe.mqbridge パッケージのクラス群

クラス名またはインターフェース名	目的
MQeCharacteristicLabels	MQSeries-Bridge コード内で使用する MQeFields オブジェクトのすべてのラベル を 1 つのグループにまとめます。
MQeClientConnectionAdminMsg	MQeClientConnection オブジェクトに対して実行する管理コマンドをカプセル化するために使用します。
MQeListenerAdminMsg	MQeListener オブジェクトに対して実行する管理コマンドをカプセル化するために使用します。
MQeMQBridgeAdminMsg	MQeMQBridge オブジェクトに対して実行する管理コマンドをカプセル化するために使用します。
MQeMQBridgeQueue	このキューは、MQSeries-Bridge に対するインターフェースとして使用します。
MQeMQBridgeQueueAdminMsg	MQSeries-Bridge キューを管理するために使用します。
MQeMQBridges	特定の MQSeries Everyplace サーバーに関連付けられているすべての MQeMQBrIdge オブジェクトをロードして保守します。
MQeMQBridgesAdminMsg	MQeBridges オブジェクトに対して実行する管理コマンドをカプセル化するために使用します。
MQeMQQMgrProxyAdminMsg	MQeQMgrProxy オブジェクトに対して実行する管理コマンドをカプセル化するために使用します。
MQeRunState	管理対象オブジェクトの*実行状態* を保持します。
MQeTransformerInterface	MQMessages を MQeMsgObject に (またはその逆に) 変換できるすべてのクラスは、このインターフェースに準拠しなければなりません。

MQeCharacteristicLabels

このクラスは、 MQSeries-Bridge コード内で使用する MQeFields オブジェクトのすべてのラベル を 1 つのグループにまとめます。

これらのラベルは、管理メッセージ、管理対象オブジェクトの実行時特性、永続レジストリー項目オブジェクトによって使用されます。

パッケージ **com.ibm.mqe.mqbridge**

このクラスは、**Object** の拡張クラスです。

定数と変数

MQeCharacteristicLabels には、以下の定数と変数が用意されています。

MQE_FIELD_LABEL_ADMINISTERED_OBJECT_CLASS

構文

```
public static final String MQE_FIELD_LABEL_ADMINISTERED_OBJECT_CLASS
```

説明 管理対象オブジェクトのすべてのレジストリー項目によって使用されます。管理対象オブジェクトをインスタンス化したいときにロードするクラスを示します。

例

```
String adminClass = fields.getUnicode(
      MQeCharacteristicLabels.MQE_FIELD_LABEL_ADMINISTERED_OBJECT_CLASS);
```

MQE_FIELD_LABEL_AFFECT_CHILDREN

構文

```
public static final String MQE_FIELD_LABEL_AFFECT_CHILDREN
```

説明 管理対象オブジェクトに対して開始アクションや削除アクションを実行したときに、子オブジェクトにも影響が及ぶかどうかを設定するために使用します。

例

```
myFields.putBoolean(
      MQeCharacteristicLabels.MQE_FIELD_LABEL_AFFECT_CHILDREN, true );
```

MQE_FIELD_LABEL_BRIDGE_NAME

構文

```
public static final String MQE_FIELD_LABEL_BRIDGE_NAME
```

説明

例

```
myFields.putUnicode(MQeCharacteristicLabels.MQE_FIELD_LABEL_BRIDGE_NAME,
                    "MQBridgeV100");
```

MQE_FIELD_LABEL_CCSID

構文

```
public static final String MQE_FIELD_LABEL_CCSID
```

説明 基礎となる MQSeries クラス Java クライアント・チャネルで使用する CCSID を保持するフィールド用のラベル。

例

```
fields.putInt( MQeCharacteristicLabels.MQE_FIELD_LABEL_CCSID , 819 );
```

MQE_FIELD_LABEL_CHILD

構文

```
public static final String MQE_FIELD_LABEL_CHILD
```

説明 管理対象オブジェクトの子オブジェクトの名前を保持する MQSeries Everyplace フィールドのラベル。

例

```
String childName = myFields.getUnicode(
       MQeCharacteristicLabels.MQE_FIELD_LABEL_CHILD );
```

MQE_FIELD_LABEL_CHILDREN

構文

```
public static final String MQE_FIELD_LABEL_CHILDREN
```

説明 MQSeries Everyplace フィールドの子オブジェクトに関連付けられているラベル。フィールドの値として、プロキシー・オブジェクト名のリストを保持します。

例

```
MQeFields[] children = bridgesDetails.getFieldsArray(
      MQeCharacteristicLabels.MQE_FIELD_LABEL_CHILDREN );
```

MQE_FIELD_LABEL_CLIENT_CONNECTION_NAME

構文

```
public static final String MQE_FIELD_LABEL_CLIENT_CONNECTION_NAME
```

説明 管理対象メッセージの送信先であるクライアント接続の名前を保持する MQSeries Everyplace フィールド用のラベル。

例

```
fields.putUnicode(
     MQeCharacteristicLabels.MQE_FIELD_LABEL_CLIENT_CONNECTION_NAME,
    "SYSTEM.DEF.SVRCONN");
```

MQE_FIELD_LABEL_DEAD_LETTER_Q_NAME

構文

```
public static final String MQE_FIELD_LABEL_DEAD_LETTER_Q_NAME
```

説明 MQSeries システム上の送達不能キューの名前を保持するフィールド用のラベル。

例

```
fields.putUnicode(
     MQeCharacteristicLabels.MQE_FIELD_LABEL_DEAD_LETTER_Q_NAME,
     "MQE.DLQ" );
```

MQE_FIELD_LABEL_DEFAULT_TRANSFORMER

構文

```
public static final String MQE_FIELD_LABEL_DEFAULT_TRANSFORMER
```

説明 メッセージ・フォーマットを MQSeries Everyplace から MQSeries へ、または MQSeries から MQSeries Everyplace へ変換するための、デフォルトのトランスフォーマー・クラスの名前を保持する MQSeries Everyplace フィールドの名前を保持するラベル。この値は、ターゲット・キュー定義でトランスフォーマーが指定されていない場合に使用されます。

例

```
fields.putUnicode(
     MQeCharacteristicLabels.MQE_FIELD_LABEL_DEFAULT_TRANSFORMER,
     "com.ibm.mqe.mqbridge.MQeBaseTransformer" );
```

MQE_FIELD_LABEL_FLOWS_PER_COMMIT

構文

```
public static final String MQE_FIELD_LABEL_FLOWS_PER_COMMIT
```

説明 MQSeries システム上の同期キューがクリーンアップされるまでに、リスナー・フローによってリスナーが何回まで送られるかを示すラベル。この値は、リスナーが MQSeries 同期キューを状態ストアとして使用している場合にのみ有効です。

例

```
fields.putInt(
     MQeCharacteristicLabels.MQE_FIELD_LABEL_FLOWS_PER_COMMIT , 100 );
```

MQE_FIELD_LABEL_HEARTBEAT_INTERVAL

構文

```
public static final String MQE_FIELD_LABEL_HEARTBEAT_INTERVAL
```

説明 MQeHeart クラスから届く各 ″ハートビート″ イベントの時間の長さを示す、時間間隔 (分単位) の値を保持するラベル。すべてのタイマー・メカニズムでは、タイマーの「刻み」としてハートビートを使用しているので、このラベルは MQSeries-Bridge 内のあらゆるタイマー・メカニズムの正確さに影響を与えます。

例

```
fields.putInt(
      MQeCharacteristicLabels.MQE_FIELD_LABEL_HEARTBEAT_INTERVAL, 5);
```

MQE_FIELD_LABEL_HOST_NAME

構文

```
public static final String MQE_FIELD_LABEL_HOST_NAME
```

説明 ホスト名フィールド用の MQSeries Everyplace フィールド・ラベル。

例

```
// Using an IP address
fields.putUnicode(MQeCharacteristicLabels.MQE_FIELD_LABEL_HOST_NAME,
                  "127.0.0.1");
// Using a host in the default domain
fields.putUnicode(MQeCharacteristicLabels.MQE_FIELD_LABEL_HOST_NAME,
                  "lizzie");
// Using a fully qualified hostname
fields.putUnicode(MQeCharacteristicLabels.MQE_FIELD_LABEL_HOST_NAME,
                  "daytona2.hursley.ibm.com");
// Using the MQSeries Java Bindings on localhost
fields.putUnicode(MQeCharacteristicLabels.MQE_FIELD_LABEL_HOST_NAME,
                  "");
```

MQE_FIELD_LABEL_LISTENER_NAME

構文

```
public static final String MQE_FIELD_LABEL_LISTENER_NAME
```

説明 MQSeries 送信キュー・リスナー名 (MQSeries 上の送信キューの名前) を保持するフィールド用のラベル。

例

```
fields.putUnicode( MQeCharacteristicLabels.MQE_FIELD_LABEL_LISTENER_NAME,
                  "MQE.XMITQ" );
```

MQE_FIELD_LABEL_LISTENER_STATE_STORE_ADAPTER

構文

```
public static final String MQE_FIELD_LABEL_LISTENER_STATE_STORE_ADAPTER
```

説明 リスナーが状態を保管するために使用するアダプター・クラスの名前を保持する MQSeries Everyplace フィールドの名前を保持するラベル。

例

```
fields.putUnicode(
      MQeCharacteristicLabels.MQE_FIELD_LABEL_LISTENER_STATE_STORE_ADAPTER,
      "com.ibm.mqe.adapters.MQeDiskFieldsAdapter" );
```

MQE_FIELD_LABEL_MAX_CONNECTION_IDLE_TIME

構文

```
public static final String MQE_FIELD_LABEL_MAX_CONNECTION_IDLE_TIME
```

説明 MQSeries クライアント接続をアイドル状態のまま開いておく最大時間の値を保持するフィールド用のラベル。この時間を超えてもアイドル状態のままになっている接続は、 MQSeries-Bridge によって閉じられます。

例

```
fields.putInt(
      MQeCharacteristicLabels.MQE_FIELD_LABEL_MAX_CONNECTION_IDLE_TIME , 5 );
```

MQE_FIELD_LABEL_MQ_BRIDGE_ADAPTER_CLASS

構文

```
public static final String MQE_FIELD_LABEL_MQ_BRIDGE_ADAPTER_CLASS
```

説明 MQSeries-Bridge アダプター・クラスの名前を保持するフィールド用のラベル。 MQSeries-Bridge アダプターは、MQSeries-Bridge ブリッジ・キューに送られたメッセージを MQSeries システムに移動するための Java クラスです。

例

```
fields.putUnicode(
      MQeCharacteristicLabels.MQE_FIELD_LABEL_MQ_BRIDGE_ADAPTER_CLASS,
      "com.ibm.mqe.mqbridge.MQeMQAdapter" );
```

MQE_FIELD_LABEL_MQ_Q_MGR_PROXY_NAME

構文

```
public static final String MQE_FIELD_LABEL_MQ_Q_MGR_PROXY_NAME
```

説明 MQSeries キュー・マネージャー・プロキシー・オブジェクトの名前を保持する MQSeries Everyplace フィールドのラベル。

例

```
String name = myFields.getUnicode(
      MQeCharacteristicLabels.MQE_FIELD_LABEL_MQ_Q_MGR_PROXY_NAME);
```

MQE_FIELD_LABEL_NAME

構文

```
public static final String MQE_FIELD_LABEL_NAME
```

説明 MQSeries-Bridge、プロキシー、クライアント接続、リスナーのいずれかの名前を保持する MQSeries Everyplace フィールドの名前を保持するラベル。

例

```
String fieldName = myFields.getUnicode(
      MQeCharacteristicLabels.MQE_FIELD_LABEL_NAME );
```

MQE_FIELD_LABEL_PASSWORD

構文

```
public static final String MQE_FIELD_LABEL_PASSWORD
```

説明 MQSeries-Bridge が MQSeries と通信するときに、ユーザー ID と一緒に使用するパスワードを保持するラベル。

例

```
fields.putUnicode(
     MQeCharacteristicLabels.MQE_FIELD_LABEL_PASSWORD, "chocolate" );
```

MQE_FIELD_LABEL_PORT

構文

```
public static final String MQE_FIELD_LABEL_PORT
```

説明 MQSeries チャネル・リスナーのポート番号を保持する MQSeries Everyplace フィールド用のラベル。

例

```
fields.putInt( MQeCharacteristicLabels.MQE_FIELD_LABEL_PORT , 1414 );
```

MQE_FIELD_LABEL_RECEIVE_EXIT

構文

```
Public static final String MQE_FIELD_LABEL_RECEIVE_EXIT
```

説明 基礎となる MQSeries クラス Java クライアント・チャネルで使用する受信出口を保持するフィールド用のラベル。

例

```
fields.putUnicode( MQeCharacteristicLabels.MQE_FIELD_LABEL_RECEIVE_EXIT,
                   "my.ReceiveExit" );
```

MQE_FIELD_LABEL_RUN_STATE

構文

```
public static final String MQE_FIELD_LABEL_RUN_STATE
```

説明 管理対象オブジェクトの現在の状態のスナップショットを保持する MQSeries Everyplace フィールドのラベル。

例

```
fields.putInt( MQeCharacteristicLabels.MQE_FIELD_LABEL_RUN_STATE,
               MQeRunState.RUN_STATE_RUNNING );
```

MQE_FIELD_LABEL_SECONDS_WAIT_FOR_MSG

構文

```
public static final String MQE_FIELD_LABEL_SECONDS_WAIT_FOR_MSG
```

説明 MQSeries 送信キュー・リスナーが MQSeries クラス Java クライアントの **GetMessage(wait)** メソッドで使用する秒数を保持するフィールド用のラベル。 (開発用のみ。)

例

```
fields.putInt(
     MQeCharacteristicLabels.MQE_FIELD_LABEL_SECONDS_WAIT_FOR_MSG, 10 );
```

MQE_FIELD_LABEL_SECURITY_EXIT

構文

```
public static final String MQE_FIELD_LABEL_SECURITY_EXIT
```

説明 基礎となる MQSeriesクラス Java クライアント・チャネルで使用するセキュリティー出口を保持するフィールド用のラベル。

例

```
fields.putUnicode( MQeCharacteristicLabels.MQE_FIELD_LABEL_SECURITY_EXIT,
                   "my.SecurityExit" );
```

MQE_FIELD_LABEL_SEND_EXIT

構文

```
public static final String MQE_FIELD_LABEL_SEND_EXIT
```

説明 基礎となる MQSeries クラス Java クライアント・チャネルで使用する送信出口を保持するフィールド用のラベル。

例

```
fields.putUnicode( MQeCharacteristicLabels.MQE_FIELD_LABEL_SEND_EXIT,
                   "my.SendExit" );
```

MQE_FIELD_LABEL_STARTUP_RULE_CLASS

構文

```
public static final String MQE_FIELD_LABEL_STARTUP_RULE_CLASS
```

説明 管理対象オブジェクトのすべてのレジストリー項目によって使用されます。管理対象オブジェクトをロード時に開始するかどうかを指定するために使用するクラスを示します。

例

```
String ruleClass = fields.getUnicode(
      MQeCharacteristicLabels.MQE_FIELD_LABEL_STARTUP_RULE_CLASS);
```

MQE_FIELD_LABEL_SYNC_Q_NAME

構文

```
public static final String MQE_FIELD_LABEL_SYNC_Q_NAME
```

説明 *SyncQName* フィールド用の MQSeries Everyplace フィールド・ラベル。想定されている送達フローの進行状況を監視するために使用します。

例

```
fields.putUnicode( MQeCharacteristicLabels.MQE_FIELD_LABEL_SYNC_Q_NAME,
                   "SYNC.Q" );
```

MQE_FIELD_LABEL_SYNC_Q_PURGE_INTERVAL

構文

```
public static final String MQE_FIELD_LABEL_SYNC_Q_PURGE_INTERVAL
```

説明 同期キューのパージを実行する時間間隔。

例

```
fields.putInt(
      MQeCharacteristicLabels.MQE_FIELD_LABEL_SYNC_Q_PURGE_INTERVAL , 60 );
```

MQE_FIELD_LABEL_SYNC_Q_PURGER_RULES_CLASS

構文

```
public static final String MQE_FIELD_LABEL_SYNC_Q_PURGER_RULES_CLASS
```

説明 *SyncQPurgerRulesClass* フィールド用の MQSeries Everyplace フィールド・ラベル。

例

```
fields.putUnicode(
      MQeCharacteristicLabels.MQE_FIELD_LABEL_SYNC_Q_PURGER_RULES_CLASS,
      "com.ibm.mqe.mqbridge.MQeSyncQueuePurgerRule" );
```

MQE_FIELD_LABEL_TRANSFORMER

構文

```
public static final String MQE_FIELD_LABEL_TRANSFORMER
```

説明 MQSeries メッセージを MQSeries Everyplace ネットワークに送る前に MQSeries Everyplace メッセージに変換するために、どのトランスフォーマー・クラスを使用するかを示すラベル。

例

```
fields.putUnicode( MQeCharacteristicLabels.MQE_FIELD_LABEL_TRANSFORMER,
                   "com.ibm.mqe.mqbridge.MQeBaseTransformer" );
```

MQE_FIELD_LABEL_UNDELIVERED_MESSAGE_RULE_CLASS

構文

```
public static final String MQE_FIELD_LABEL_UNDELIVERED_MESSAGE_RULE_CLASS
```

説明 メッセージを宛先に送信できなかった場合に使用するルールの Java クラス名を保持するフィールド用のラベル。

例

```
fields.putUnicode(
      MQeCharacteristicLabels.MQE_FIELD_LABEL_UNDELIVERED_MESSAGE_RULE_CLASS,
      "com.ibm.mqe.mqbridge.MQeUndeliveredMessageRule" );
```

MQE_FIELD_LABEL_USER_ID

構文

```
public static final String MQE_FIELD_LABEL_USER_ID
```

説明 MQSeries-Bridge が MQSeries と通信するときに使用するユーザー ID 用のラベル。

例

```
fields.putUnicode( MQeCharacteristicLabels.MQE_FIELD_LABEL_USER_ID, "mqm" );
```

コンストラクター

MQeCharacteristicLabels

構文

```
public MQeCharacteristicLabels()
```

説明 MQeCharacteristicLabels オブジェクトを作成します。

パラメーター

なし

戻り値 なし

例外 なし

MQeClientConnectionAdminMsg

これは、管理コマンドをカプセル化するための特殊な MQSeries Everyplace メッセージです。このメッセージは、管理を担当しているアプリケーションによって生成されます。

このクラスには、ターゲット MQSeries Everyplace システムで実行されるロジックも入っています。

管理キューは、**performAction** メソッドを呼び出します。

パッケージ **com.ibm.mqe.mqbridge**

このクラスは、 MQeMQQMgrProxyAdminMsg の拡張クラスです。

定数と変数

MQeQueueAdminMsg には、 MQeMQQMgrProxyAdminMsg の定数と変数の他に、以下の定数と変数が用意されています。

コンストラクター

MQeClientConnectionAdminMsg

構文

1. `public MQeClientConnectionAdminMsg() throws Exception`
2. ```
   public MQeClientConnectionAdminMsg(String bridgeName,
                                      String nameOfMQQMgrProxy,
                                      String clientConnectionName,
                                      boolean affectChildren)
                                                  throws Exception
   ```

説明 2 つのコンストラクターがあります。

1. このバージョンは、デフォルトの MQeClientConnectionAdminMsg を作成して初期化します。
2. このバージョンは、管理メッセージを初期化するために必要なフィールドを組み込みます。ただし、管理メッセージが保持するアクションは組み込みません。

パラメーター

bridgeName 管理メッセージの送信先である MQSeries-Bridge の名前を含んだストリング。ヌルまたは "" に設定した場合は、何も設定しなかったことになります。
   ```

nameOfMQQMgrProxy
管理メッセージの送信先であるプロキシーの名前を含んだストリング。ヌルまたは "" に設定した場合は、何も設定しなかったことになります。

clientConnectionName
管理メッセージの送信先であるクライアント接続の名前を含んだストリング。ヌルまたは "" に設定した場合は、何も設定しなかったことになります。

affectChildren この管理メッセージがすべての子オブジェクトに影響を及ぼすかどうかを設定するためのブール・フラグ。**開始**アクションと**削除**アクションのいずれかの場合にのみ適用されます。

戻り値 なし

例外 いずれかのパラメーターに無効文字が含まれている場合は失敗

例

```
MQeClientConnectionAdminMsg msg ;
msg = new MQeClientConnectionAdminMsg( "ExampleQM.MQBridgeV100"
                                       , "MQA"
                                       , "MQ.to.ExampleQM"
                                       , false
                                       );
```

メソッド

メソッド	目的
characteristics	このタイプの管理メッセージで必要なすべてのフィールドを含んだ MQeFields オブジェクトを作成します。
getClientConnectionName	管理対象オブジェクトからクライアント接続名を取得します。
getName	管理対象になるオブジェクトの名前を取得します。
putClientConnectionName	MQeFields 管理メッセージ・オブジェクト内のフィールドにクライアント接続名を入れます。
setName	MQeFields 管理メッセージ・オブジェクト内のフィールドに名前情報を入れたり、この MQSeries-Bridge に関連付けられている MQSeries Everyplace キュー・マネージャーの名前を設定したりします。

MQeClientConnectionAdminMsg characteristics

構文

```
public MQeFields characteristics() throws Exception
```

説明 このタイプの管理メッセージで必要なすべてのフィールドを含んだ MQefields オブジェクトを作成します。

MQeMQQMgrProxyAdminMsg クラスの **characteristics()** をオーバーライドします。

パラメーター

なし

戻り値 このリソースの特性を含んだ MQeFields オブジェクト。生成されるフィールド・オブジェクトから、このリソースのフィールド名とフィールド・タイプの完全セットを判別できます。

例外

java.lang.Exception	MQeFields オブジェクトを作成できなかった場合

例

```
MQeClientConnectionAdminMsg msg;
msg = new MQeClientConnectionAdminMsg( "ExampleQM.MQBridgeV100",
                                       "MQA",
                                       "MQ.to.ExampleQM",
                                       false);
MQeFields cconAdminCharacteristics = msg.characteristics();
```

MQeClientConnectionAdminMsg getClientConnectionName

構文

```
public String getClientConnectionName() throws Exception
```

説明 管理対象オブジェクトからクライアント接続名を取得します。このメソッドは、 MQeClientConnectionAdminMsg またはそのいずれかの下位オブジェクトに対して実行できます。

パラメーター

なし

戻り値 この管理メッセージの送信先であるクライアント接続の名前

例外

java.lang.Exception	この管理メッセージにその名前が設定されていない場合、または設定されている名前が無効である場合

例

```
MQeClientConnectionAdminMsg msg;
msg = new MQeClientConnectionAdminMsg( "ExampleQM.MQBridgeV100",
                                       "MQA",
```

```
                                          "MQ.to.ExampleQM",
                                          false);
String cconName = msg.getClientConnectionName();
```

MQeClientConnectionAdminMsg getName

構文

```
public String getName()
```

説明 管理対象になるクライアント接続の名前を取得します。この場合は、**setName()** メソッドまたは **putClientConnectionName()** メソッドによって設定されているクライアント接続の名前です。このクラスのオブジェクトに対して実行する場合は、 `getClientConnectionName()` と同じ結果になります。

MQeMQQMgrProxyAdminMsg クラスの **getName()** をオーバーライドします。

パラメーター

なし

戻り値 作成する管理対象オブジェクトの名前を含んだストリング、または名前が設定されていない場合はヌル

例外 なし

例

```
MQeClientConnectionAdminMsg msg;
msg = new MQeClientConnectionAdminMsg( "ExampleQM.MQBridgeV100",
                                          "MQA",
                                          "MQ.to.ExampleQM",
                                          false);
String cconName = msg.getName();
```

MQeClientConnectionAdminMsg putClientConnectionName

構文

```
public void putClientConnectionName(String clientConnectionName)
                                                        throws Exception
```

説明

管理メッセージに MQSeries キュー・マネージャー名を追加するために使用します。 MQSeries Everyplace フィールド管理メッセージ・オブジェクト内の MQSeries Everyplace フィールドにクライアント接続名を入れます。

パラメーター

clientConnectionName
管理メッセージの送信先であるクライアント接続の名前を含んだストリング。このストリングに有効な文字だけが含まれ

ていることを確かめるには、 **validateName()** メソッドを使って妥当性検査を実行します。

戻り値 なし

例外

java.lang.Exception
名前パラメーターに無効文字がある場合

例

```
MQeClientConnectionAdminMsg msg = new MQeClientConnectionAdminMsg();
msg.setName( "ExampleQM.MQBridgeV100" ,
             "MQA" ,
             "MQ.to.ExampleQM" );
```

MQeClientConnectionAdminMsg setName

構文

```
public void setName(String bridgeName,
                    String mqQMgrProxyName,
                    String clientConnectionName) throws Exception
```

説明 MQSeries Everyplace フィールド管理メッセージ・オブジェクト内の MQSeries Everyplace フィールドに名前情報を入れたり、この MQSeries-Bridge に関連付けられている MQSeries Everyplace キュー・マネージャーの名前を設定したりします。

パラメーター

bridgeName 管理メッセージの送信先である MQSeries-Bridge の名前を含んだストリング。ヌルまたは "" に設定した場合は、何も設定しなかったことになります。

mqQMgrProxyName
管理メッセージの送信先であるプロキシーの名前を含んだストリング。ヌルまたは "" に設定した場合は、何も設定しなかったことになります。

clientConnectionName
管理メッセージの送信先であるクライアント接続の名前を含んだストリング。ヌルまたは "" に設定した場合は、何も設定しなかったことになります。

戻り値 なし

例外

java.lang.Exception 名前パラメーターに無効文字がある場合

例

```
MQeClientConnectionAdminMsg msg = new MQeClientConnectionAdminMsg();
msg.setName( "ExampleQM.MQBridgeV100" ,
             "MQA" ,
             "MQ.to.ExampleQM" )
```

MQeListenerAdminMsg

これは、管理コマンドをカプセル化するための特殊な MQSeries Everyplace メッセージです。このメッセージは、管理を担当しているアプリケーションによって生成されます。

このクラスには、ターゲット MQSeries Everyplace システムで実行されるロジックも入っています。

管理キューは、**performAction()** メソッドを呼び出します。

パッケージ **com.ibm.mqe.mqbridge**

このクラスは、 MQeClientConnectionAdminMsg の拡張クラスです。

コンストラクター

MQeListenerAdminMsg

構文

```
1. public MQeListenerAdminMsg() throws Exception
2. public MQeListenerAdminMsg(String bridge,
                              String MQQMgrProxy,
                              String clientConnection,
                              String listener
                              boolean affectChildren) throws Exception
```

説明 2 つのコンストラクターがあります。

1. このバージョンは、デフォルトの MQeListenerAdminMsg を作成して初期化します。
2. このバージョンは、MQSeries Everyplace キュー・マネージャーの名前、MQSeries-Bridge の名前、プロキシーの名前、クライアント接続の名前、リスナーの名前を組み込みます。

パラメーター

bridge 管理メッセージの送信先である MQSeries-Bridge の名前を含んだストリング。ヌルまたは "" に設定した場合は、何も設定しなかったことになります。

MQQMgrProxy リスナーの読み込み先として設定されている送信キューを所有する MQSeries キュー・マネージャーの名前を含んだストリング。ヌルまたは "" に設定した場合は、何も設定しなかったことになります。

clientConnection
MQSeries キュー・マネージャーと通信するためのクライアン

ト接続の名前を含んだストリング。ヌルまたは "" に設定した場合は、何も設定しなかったことになります。

listener リスナーの名前を含んだストリング。メッセージを MQSeries Everyplace ネットワークに移動できる状態にするには、この名前を、リスナーの "listen" 先の MQSeries 上の送信キューの名前と一致させます。

affectChildren この管理メッセージがリスナーの子オブジェクトに影響を及ぼすかどうかを設定するためのブール・フラグ。

戻り値 なし

例外 いずれかのパラメーターに無効文字が含まれている場合は失敗

例

```
MQeListenerAdminMsg msg = new MQeListenerAdminMsg( "MQBridgeV100",
                                                   "lizzieQM",
                                                   "svrconn",
                                                   "MQE.XMITQ",
                                                   true);
```

メソッド

メソッド	目的
characteristics	このタイプの管理メッセージで必要なすべてのフィールドを含んだ MQeFields オブジェクトを作成します。
getListenerName	管理対象オブジェクトからリスナー名を取得します。
getName	作成する管理対象オブジェクトの名前を取得します。
putListenerName	MQeFields 管理メッセージ・オブジェクト内のフィールドにリスナー名を入れます。
setName	MQeFields 管理メッセージ・オブジェクト内のフィールドに名前情報を入れたり、この MQSeries-Bridge に関連付けられている MQSeries Everyplace キュー・マネージャーの名前を設定したりします。

MQeListenerAdminMsg characteristics

構文

```
public MQeFields characteristics() throws Exception
```

説明 このタイプの管理メッセージで必要なすべてのフィールドを含んだ MQeFields オブジェクトを作成します。

リソースの特性の入った MQeFields オブジェクトを戻します。生成されるフィールド・オブジェクトから、このリソースのフィールド名とフィールド・タイプの完全セットを判別できます。

MQeClientConnectionAdminMsg クラスの **characteristics()** をオーバーライドします。

パラメーター
なし

戻り値 このリソースの特性を含んだ MQeFields オブジェクト。

例外

java.lang.Exception	MQeFields オブジェクトを作成できなかった場合

例

```
MQeListenerAdminMsg msg = new MQeListenerAdminMsg( "MQBridgeV100",
                                                   "lizzieQM",
                                                   "svrconn",
                                                   "MQE.XMITQ",
                                                   true );
MQeFields characteristics = msg.characteristics();
```

MQeListenerAdminMsg getListenerName

構文

```
public String getListenerName() throws Exception
```

説明 管理対象オブジェクトからリスナー名を取得します。

MQeListenerAdminMsg オブジェクトに対してのみ実行できます。

パラメーター
なし

戻り値 リスナーの名前。

例外

java.lang.Exception
この管理メッセージにその名前が設定されていない場合、または設定されている名前が無効である場合

例

```
MQeListenerAdminMsg msg = new MQeListenerAdminMsg( "MQBridgeV100",
                                                   "lizzieQM",
                                                   "svrconn",
                                                   "MQE.XMITQ",
                                                   true);
String listenerName = msg.getListenerName();
```

MQeListenerAdminMsg getName

構文

```
public String getName()
```

説明 管理対象になるクライアント接続の名前を取得します。

このクラスのオブジェクトに対して実行する場合は、 **getListenerName()** と同じ結果になります。

MQeClientConnectionAdminMsg クラスの **getName()** をオーバーライドします。

パラメーター

なし

戻り値 作成する管理対象オブジェクトの名前を含んだストリング、または名前が設定されていない場合はヌル。

例外 なし

例

```
MQeListenerAdminMsg msg = new MQeListenerAdminMsg( "MQBridgeV100",
                                                   "lizzieQM",
                                                   "svrconn",
                                                   "MQE.XMITQ",
                                                   true);
String listenerName = msg.getName();
```

MQeListenerAdminMsg putListenerName

構文

```
public void putListenerName(String listener) throws Exception
```

説明 管理メッセージに MQSeries キュー・マネージャー名を追加するために使用されます。 MQeFields 管理メッセージ・オブジェクト内のフィールドにリスナー名を入れます。

このメソッドは、管理メッセージの送信元によって使用されます。

パラメーター

listener リスナーの名前を含んだストリング。メッセージを MQSeries Everyplace ネットワークに移動できる状態にするには、この名前を、リスナーの ″listen″ 先の MQSeries 上の送信キューの名前と一致させます。

戻り値 なし

例外

java.lang.Exception 名前パラメーターに無効文字がある場合

例

```
MQeListenerAdminMsg msg = new MQeListenerAdminMsg();
msg.putListenerName("MQE.XMITQ");
```

MQeListenerAdminMsg setName

構文

```
public void setName(String bridge,
                    String mqQMgrProxy,
                    String clientConnection
                    String listener) throws Exception
```

説明 MQeFields 管理メッセージ・オブジェクト内のフィールドに名前情報を入れたり、この MQSeries-Bridge に関連付けられている MQSeries Everyplace キュー・マネージャーの名前を設定したりします。

このメソッドは、管理メッセージの送信元によって使用されます。

パラメーター

bridge 管理メッセージの送信先である MQSeries-Bridge の名前を含んだストリング。ヌルまたは "" に設定した場合は、何も設定しなかったことになります。

MQQMgrProxy リスナーの読み込み先として設定されている送信キューを所有する MQSeries キュー・マネージャーの名前を含んだストリング。ヌルまたは "" に設定した場合は、何も設定しなかったことになります。

clientConnection
MQSeries キュー・マネージャーと通信するためのクライアント接続の名前を含んだストリング。ヌルまたは "" に設定した場合は、何も設定しなかったことになります。

listener リスナーの名前を含んだストリング。メッセージを MQSeries Everyplace ネットワークに移動できる状態にするには、この名前を、リスナーの "listen" 先の MQSeries 上の送信キューの名前と一致させます。

戻り値 なし

例外

java.lang.Exception
名前パラメーターに無効文字がある場合

例

```
MQeListenerAdminMsg msg = new MQeListenerAdminMsg();
msg.setName("MQBridgeV100", "lizzieQM", "svrconn", "MQE.XMITQ");
```

MQeMQBridgeAdminMsg

これは、管理コマンドをカプセル化するための特殊な MQSeries Everyplace メッセージです。このメッセージは、管理を担当しているアプリケーションによって生成されます。

このクラスには、ターゲット MQSeries Everyplace システムで実行されるロジックも入っています。

管理キューは、**performAction()** メソッドを呼び出します。

パッケージ **com.ibm.mqe.mqbridge**

このクラスは、 MQeMQBridgesAdminMsg の拡張クラスです。

定数と変数

MQeQueueAdminMsg には、 MQeMQBridgesAdminMsg の定数と変数の他に、以下の定数と変数が用意されています。

DEFAULT_MQBRIDGE_NAME

```
public static final String DEFAULT_MQBRIDGE_NAME
```

コンストラクター

MQeMQBridgeAdminMsg

構文

1. `public MQeMQBridgeAdminMsg() throws Exception`
2. `public MQeMQBridgeAdminMsg(String bridgeName,`
   `boolean affectChildren) throws Exception`

説明 2 つのコンストラクターがあります。

1. このバージョンは、デフォルトの MQeBridgeAdminMsg を作成して初期化します。
2. このバージョンは、 MQSeries-Bridge の名前と、管理コマンドによって子オブジェクトが影響を受けるかどうかを指定するフラグを組み込みます。

パラメーター

bridge 管理メッセージの送信先である MQSeries-Bridge の名前を含んだストリング。ヌルまたは "" に設定した場合は、何も設定しなかったことになります。

affectChildren この管理メッセージがリスナーの子オブジェクトに影響を及ぼすかどうかを設定するためのブール・フラグ。

戻り値 なし

例外

java.lang.Exception	いずれかのパラメーターに無効文字が含まれている場合

例

```
1. MQeMQBridgeAdminMsg msg = new MQeMQBridgeAdminMsg();
2. public MQeMQBridgeAdminMsg(java.lang.String bridge,
                              boolean affectChildren) Exception
```

メソッド

メソッド	目的
characteristics	このタイプの管理メッセージで必要なすべてのフィールドを含んだ MQeFields オブジェクトを作成します。
create	この管理メッセージを「作成」メッセージにします。
delete	この管理メッセージを「削除」メッセージにします。
getBridgeName	管理対象オブジェクトから MQSeries-Bridge 名を取得します。
getName	作成する管理対象オブジェクトの名前を取得します。
putBridgeName	MQeFields 管理メッセージ・オブジェクト内のフィールドに MQSeries-Bridge 接続名を入れます。
setName	MQeFields 管理メッセージ・オブジェクト内のフィールドに名前情報を入れたり、この MQSeries-Bridge に関連付けられている MQSeries Everyplace キュー・マネージャーの名前を設定したりします。

MQeMQBridgeAdminMsg characteristics

構文

```
public MQeFields characteristics() throws Exception
```

説明 このタイプの管理メッセージで必要なすべてのフィールドを含んだ MQeFields オブジェクトを作成します。

このリソースの特性を含んだフィールド・オブジェクトを戻します。生成されるフィールド・オブジェクトから、このリソースのフィールド名とフィールド・タイプの完全セットを判別できます。

MQeBridgesAdminMsg クラスの **characteristics()** をオーバーライドします。

パラメーター

なし

戻り値 このリソースの特性を含んだ MQeFields オブジェクト。

例外

java.lang.Exception MQeFields オブジェクトを作成できなかった場合

例

```
MQeMQBridgeAdminMsg msg = new MQeMQBridgeAdminMsg("MQBridgeV100", true);
MQeFields bridgeCharacteristics = msg.characteristics();
```

MQeMQBridgeAdminMsg create

構文

```
public void create(MQeFields parms) throws Exception
```

説明

管理メッセージの送信元によって使用されます。

この管理メッセージを「作成」メッセージにします。ターゲット MQSeries Everyplace システムがこのメッセージを処理すると、 MQSeries キュー・マネージャー項目が作成されます。

MQeAdminMsg クラスの **create()** をオーバーライドします。

パラメーター

parms メッセージに追加する任意のエクストラ・パラメーター、またはヌル。

戻り値 なし

例外

java.lang.Exception MQeFields オブジェクトを作成できなかった場合

例

```
// Form a message that will create a new MQBridge.
MQeMQBridgeAdminMsg msg = new MQeMQBridgeAdminMsg();
msg.create( new MQeFields() );
```

MQeMQBridgeAdminMsg getBridgeName

構文

```
public String getBridgeName() throws Exception
```

説明

管理対象オブジェクトから MQSeries-Bridge 名を取得します。

MQeMQBridgeAdminMsg またはそのいずれかの下位オブジェクトに対して実行できます。

パラメーター

なし

戻り値 MQSeries-Bridge の名前。

例外

java.lang.Exception	この管理メッセージにその名前が設定されていない場合、または設定されている名前が無効である場合

例

```
MQeMQBridgeAdminMsg msg = new MQeMQBridgeAdminMsg("MQBridgeV100", true);
String bridgeName = msg.getBridgeName();
```

MQeMQBridgeAdminMsg getName

構文

```
public String getName()
```

説明 管理対象になる MQSeries-Bridge の名前を取得します。このクラスのオブジェクトに対して実行する場合は、 **getBridgeName()** と同じ結果になります。

MQeBridgesAdminMsg クラスの **getName()** をオーバーライドします。

パラメーター
なし

戻り値 作成する管理対象オブジェクトの名前を含んだストリング、または名前が設定されていない場合はヌル

例外 なし

例

```
MQeMQBridgeAdminMsg msg = new MQeMQBridgeAdminMsg("MQBridgeV100", true);
String bridgeName = msg.getName();
```

MQeMQBridgeAdminMsg delete

構文

1. ```
   public void delete(MQeFields parms,
                      boolean affectChildren) throws Exception
   ```
2. ```
   public void delete(MQeFields parms) throws Exception
   ```

説明 このメソッドには 2 つのバージョンがあります。

1. このバージョンは、管理メッセージの送信元によって使用され、この管理メッセージを「削除」メッセージにします。ターゲット MQSeries Everyplace システムがこのメッセージを処理すると、指定の MQeMQBridge オブジェクトが検出されて削除されます。この操作は、その他の MQSeries-Bridge オブジェクト・タイプから継承されます。
2. このバージョンは、 `delete((MQeFields) parms, false)` と同じ結果になります。ターゲット MQSeries Everyplace システムがこのメッセージを処理す

ると、指定の MQeMQBridge オブジェクトが検出されて削除されます。この操作は、MQSeries-Bridge オブジェクト・タイプから継承されます。

MQeAdminMsg クラスの **delete()** をオーバーライドします。

パラメーター

parms メッセージに追加する任意のエクストラ・パラメーター、またはヌル。

affectChildren この管理メッセージがリスナーの子オブジェクトに影響を及ぼすかどうかを設定するためのブール・フラグ。

戻り値 なし

例外

java.lang.Exception 削除が失敗した場合

例

```
1. // Form a message that will delete an MQBridge and its children.
   MQeMQBridgeAdminMsg msg = new MQeMQBridgeAdminMsg();
   msg.delete( new MQeFields(), true );
2. // Form a message that will delete an MQBridge.
   MQeMQBridgeAdminMsg msg = new MQeMQBridgeAdminMsg();
   msg.delete( new MQeFields() );
```

MQeMQBridgeAdminMsg putBridgeName

構文

```
public void putBridgeName(String bridge) throws Exception
```

説明

管理メッセージに MQSeries キュー・マネージャー名を追加するために、管理メッセージの送信元によって使用されます。

MQeFields 管理メッセージ・オブジェクト内のフィールドに MQSeries-Bridge 名を入れます。

パラメーター

bridge 管理メッセージの送信先である MQSeries-Bridge の名前を含んだストリング。ヌルまたは "" に設定した場合は、何も設定しなかったことになります。

戻り値 なし

例外

java.lang.Exception 名前パラメーターに無効文字がある場合

例

```
MQeMQBridgeAdminMsg msg = new MQeMQBridgeAdminMsg();
msg.putBridgeName("MQBridgeV100");
```

MQeMQBridgeAdminMsg setName

構文

```
public void setName(String bridge) throws Exception
```

説明

管理メッセージに MQSeries-Bridge を追加するために、管理メッセージの送信元によって使用されます。

MQeFields 管理メッセージ・オブジェクト内のフィールドに名前情報を入れたり、この MQSeries-Bridge に関連付けられている MQSeries Everyplace キュー・マネージャーの名前を設定したりします。

MQeAdminMsg クラスの **setName()** をオーバーライドします。

パラメーター

bridge 管理メッセージの送信先である MQSeries-Bridge の名前を含んだストリング。ヌルまたは "" に設定した場合は、何も設定しなかったことになります。

戻り値 なし

例外

java.lang.Exception 名前パラメーターに無効文字がある場合

例

```
MQeMQBridgeAdminMsg msg = new MQeMQBridgeAdminMsg();
msg.setName("MQBridgeV100");
```

MQeMQBridgeQueue

このキューは、MQSeries-Bridge に対するインターフェースとして使用するもので、ユーザー作成の「変換関数」コードに渡されます。

変換関数は、メッセージ・フォーマットを MQSeries Everyplace から MQSeries に (またはその逆に) 変換する作業を担当しますが、 MQeMQBridgeQueue クラスへの参照が変換コードに渡されるのは、メッセージが MQSeries Everyplace から MQSeries に移動する場合に限られます。

このクラスは、ユーザー作成のトランスフォーマー・クラスのインプリメンテーションが、メッセージ・データに対して変換操作を実行するときに使用できる詳細情報を保持します。

パッケージ **com.ibm.mqe.mqbridge**

このクラスは、MQeRemoteQueue の拡張クラスです。

メソッドの要約

メソッド	目的
getMQQMgr	MQSeries-Bridge キューが使用している MQSeries キュー・マネージャーの名前を取得します。
getQueueAttribute	MQSeries-Bridge キューに付加されている属性を取得します。
getQueueManagerName	所有側のキュー・マネージャー名を取得します。
getQueueName	MQSeries-Bridge キューの名前を取得します。
getRemoteQName	MQSeries-Bridge キューが参照する MQSeries キューの名前を取得します。

MQeMQBridgeQueue getMQQMgr

構文

```
public String getMQQMgr() throws Exception
```

説明 MQSeries-Bridge キューが使用している MQSeries キュー・マネージャーの名前を取得します。

パラメーター

なし

戻り値

MQSeries-Bridge キューが使用している MQSeries キュー・マネージャーの名前を含むストリング。

管理者が MQSeries-Bridge キュー・パラメーターに "" またはヌルを指定した場合、このストリングによって戻される値は、 **getQueueManagerName()** メソッドによって戻される値と一致します。

例外

java.lang.Exception	名前の取得が失敗した場合、または名前が無効である場合

例

```
String mqQMgrName = transformersBridgeQueue.getMQQMgr();
```

MQeMQBridgeQueue getQueueAttribute

構文

```
public MQeAttribute getQueueAttribute() throws Exception
```

説明

MQSeries-Bridge キューに付加されている属性を取得します。

MQeQueue クラスの **getQueueAttribute()** をオーバーライドします。

パラメーター

なし

戻り値 MQSeries-Bridge キューに設定されている属性を含んだ MQeAttribute オブジェクト。

例外

java.lang.Exception	取得が失敗した場合

例

```
MQeAttribute attribute = transformersBridgeQueue.getQueueAttribute();
```

MQeMQBridgeQueue getQueueManagerName

構文

```
public String getQueueManagerName() throws Exception
```

説明

所有側の MQSeries キュー・マネージャーの名前を取得します。

MQeQueue クラスの **getQueueManagerName()** をオーバーライドします。

パラメーター

なし

戻り値 所有側のキュー・マネージャーの名前を含んだストリング

例外

java.lang.Exception 取得が失敗した場合、または名前が無効である場合

例

```
String qMgrName = transformersBridgeQueue.getQueueManagerName();
```

MQeMQBridgeQueue getQueueName

構文

```
public String getQueueName() throws Exception
```

説明 MQSeries Everyplace 上で認識されている MQSeries-Bridge キューの名前を取得します。

MQeQueue クラスの **getQueueName()** をオーバーライドします。

パラメーター

なし

戻り値 MQSeries-Bridge キューの名前を含むストリング。

例外

java.lang.Exception 取得が失敗した場合、または名前が無効である場合

例

```
String mqQName = transformersBridgeQueue.getQueueName();
```

MQeMQBridgeQueue getRemoteQName

構文

```
public String getRemoteQName() throws Exception
```

説明 MQSeries-Bridge キューが参照する MQSeries キューの名前を取得します。

パラメーター

なし

戻り値

管理者がキューを設定したときに指定した、リモート・キュー名の構成情報のストリング内容を含んだストリング。

リモート・キュー名がブランクまたはヌルの場合、 MQSeries-Bridge キューそのもののキュー名が戻されます。

例外

java.lang.Exception 取得が失敗した場合、または名前が無効である場合

例

```
String remoteQName = transformersBridgeQueue.getRemoteQName();
```

MQeMQBridgeQueueAdminMsg

MQBridge キューを管理するために使用します。

パッケージ **com.ibm.mqe.mqbridge**

このクラスは、 MQeRemoteQueueAdminMsg の拡張クラスです。

定数と変数

MQeQueueAdminMsg には、以下の定数と変数が用意されています。

Queue_BridgeName
このキューの詳細をレジストリーにダンプするときに使用する定数。このフィールドには MQSeries-Bridge の名前が入ります。

```
public static final String Queue_BridgeName
```

Queue_ClientConnection
このキューの詳細をレジストリーにダンプするときに使用する定数。このフィールドにはクライアント接続の名前が入ります。

```
public static final String Queue_ClientConnection
```

Queue_MaxIdleTime
MQSeries-Bridge キューが接続を接続プールに戻さないで、アイドル状態のままにしておける時間を示した、 *MaxIdleTime* 構成パラメーター・フィールドの名前。

```
public static final String Queue_MaxIdleTime[]
```

Queue_MQQMgr
このキューの詳細をレジストリーにダンプするときに使用する定数。このフィールドには MQSeries キュー・マネージャーの名前が入ります。

```
public static final String Queue_MQQMgr
```

Queue_RemoteQName
このキューの詳細をレジストリーにダンプするときに使用する定数。このフィールドにはリモート・キュー名が入ります。

```
public static final String Queue_MQQMgr
```

Queue_Transformer
MQSeries Everyplace メッセージを MQSeries メッセージに変換するときに使用するトランスフォーマーの名前。

```
public static final String Queue_Transformer
```

キューの特性

Name キューの名前。

MQSeries-Bridge キューの場合は、 MQSeries Everyplace システム上で認識されている MQSeries キューの名前になります。(ASCII)

この特性は必須です。

MQeField ラベル: *MQeMQBridgeQueueAdminMsg.Admin_Name*

QMgrName

MQeQueueAdminMsg クラスで説明されています。

MQSeries-Bridge キューの場合は、キューが置かれている MQSeries キュー・マネージャーの名前を保持するはずであり、 MQSeries-Bridge が直接に接続しているキュー・マネージャーの名前であるとは限りません。

MQeField ラベル: *MQeMQBridgeQueueAdminMsg.Queue_QMgrName*

Active

CreationDate

MQeQueueAdminMsg クラスで説明されています。

MQeField ラベル: *MQeMQBridgeQueueAdminMsg.Queue_CreationDate*

説明 MQeQueueAdminMsg クラスで説明されています。

MQeField ラベル: *MQeMQBridgeQueueAdminMsg.Queue_Description*

Expiry MQeQueueAdminMsg クラスで説明されています。

メッセージの期限切れ時間。

この値は、メッセージのトランスフォーマーが、 MQSeries システムに送られる MQSeries メッセージの期限切れ時間を設定するために使用できます。

1 未満の値は、"期限切れがない" という意味になります。

ユーザーのカスタマイズによって、トランスフォーマーがこの情報を使用しないように設定することもできます。

MQSeries Everyplace メッセージそのものに、この値をオーバーライドする期限切れ時間が含まれていることもあります。

MQeField ラベル: *MQeMQBridgeQueueAdminMsg.Queue_Expiry*

MaxMsgSize

MQeQueueAdminMsg クラスで説明されています。

この最大メッセージ長は、任意指定フィールドです。これは、MQSeries-Bridge 基本コードによって使用されるものではなく、むしろ、ルール・クラスがメッセージを MQSeries キューに送るかどうかを判断するときに使用します。

この値と、この MQSeries-Bridge キューが参照する MQSeries キューによって指定されている値とが、同じかどうかを確かめるためのチェックは実行されません。たとえば、MQSeries キューでそれ以上大きなメッセージを受け入れるこ

とができるにもかかわらず、この値を使用するルールによって、一定の長さを超えるメッセージを MQSeries キューに送らないようにすることができます。

MQeField ラベル: *MQeMQBridgeQueueAdminMsg.Queue_MaxMsgSize*

Mode Type

MQeQueueAdminMsg クラスで説明されています。

MQeField ラベル: *MQeMQBridgeQueueAdminMsg.Queue_Mode*

注: Cryptor、Authenticator、Compressor (後述) の各特性は、このキューに渡されるメッセージのセキュリティー・レベルを設定するキュー属性のセットを定義します。メッセージは、最初に送られる MQSeries Everyplace 内のポイントから MQSeries-Bridge キューに渡されるポイントまで、少なくとも MQSeries Everyplace によって実施される指定のセキュリティー・レベルで保護されます。これらの値は、MQSeries-Bridge キューが MQSeries システムに対してメッセージを渡す場合には、適用されません。MQSeries に渡す場合には、 MQSeries-Bridge 上のクライアント接続構成オブジェクトで指定されている、セキュリティー出口、送信出口、受信出口によってメッセージが保護されます。 MQSeries-Bridge キューと MQSeries システムとの間のリンクに、この Cryptor クラス、Authenticator クラス、Compressor クラスで指定されているセキュリティー属性と、少なくとも同じレベルのセキュリティーが確保されているかどうかのチェックは行われません。

AttrRule

MQeQueueAdminMsg クラスで説明されています。

MQeField ラベル: MQeMQBridgeQueueAdminMsg.Queue_AttrRule

Authenticator

MQeQueueAdminMsg クラスで説明されています。

MQeField ラベル: MQeMQBridgeQueueAdminMsg.Queue_Authenticator

Compressor

MQeQueueAdminMsg クラスで説明されています。

MQeField ラベル: MQeMQBridgeQueueAdminMsg.Queue_Compressor

Cryptor

MQeQueueAdminMsg クラスで説明されています。

MQeField ラベル: MQeMQBridgeQueueAdminMsg.Queue_Cryptor

TargetRegistry

MQeQueueAdminMsg クラスで説明されています。

MQeField ラベル: MQeMQBridgeQueueAdminMsg.Queue_TargetRegistry

Rule MQeQueueAdminMsg クラスで説明されています。

MQeField ラベル: MQeMQBridgeQueueAdminMsg.Queue_Rule

Bridge Name

MQSeries ネットワークに MQSeries Everyplace メッセージを送るために使用する MQSeries-Bridge の名前。(ASCII)

このフィールドは必須です。

MQeField ラベル: MQeMQBridgeQueueAdminMsg.Queue_BridgeName

MQSeries Queue Manager Proxy

MQSeries ネットワークに MQSeries Everyplace メッセージを送るために使用する、 MQSeries-Bridge 内の MQSeries キュー・マネージャー・プロキシーの名前。

このフィールドは必須です。

(この名前は、クライアント接続によってメッセージが最初に送られる宛先となった、 MQSeries キュー・マネージャーの名前と同じです。)

MQeField ラベル: MQeMQBridgeQueueAdminMsg.Queue_ClientConnection

MQSeries Remote Q name

リモート MQSeries キュー名は任意指定です。

これは、 MQSeries キュー・マネージャー上でメッセージの宛先となっているキューの名前を指します。

ブランクまたはヌル に設定した場合は、指定のクライアント接続の相手側にある MQSeries キュー・マネージャー上のキューの名前が、この MQSeries-Bridge キュー定義と同じ名前になります (つまり、上記の ″Name″ フィールドから値が導出されるということ)。

このパラメーターは、2 つの MQSeries キュー・マネージャー上にある同じ名前の 2 つのキューに対して、 MQSeries Everyplace システムにおいてそれぞれ異なる MQSeries-Bridge キュー名を与えるための、一種の名前変更機能を実現します。

MQeField ラベル: MQeMQBridgeQueueAdminMsg.Queue_RemoteQName

Transformer class

トランスフォーマー・クラスは、 MQSeries メッセージを MQSeries ネットワークに送る前に、 MQSeries Everyplace メッセージを MQSeries メッセージに変換するための Java クラスの名前です。

このパラメーターをブランクにしておくか、ヌル に設定すると、指定した MQSeries-Bridge のデフォルトのトランスフォーマー・クラスが使用されます。

このパラメーターは任意指定です。

MQeField ラベル: MQeMQBridgeQueueAdminMsg.Queue_Transformer

Return idle connection timeout

"アイドル時間" のパラメーターでは、キューがアイドル状態の MQSeries 接続を保持しておける最大時間、つまり、 MQSeries-Bridge によって管理されているアイドル接続プールに戻さなければならなくなるまでの最大時間を、分単位で指定します。

キューがしばらくの間使用されない場合は、基礎となっている MQSeries 接続がプールに戻されるので、別のキューがその接続を検出して使用することができます。アイドル状態になっていた元のキューにメッセージ活動が発生すると、そのキューは、プールの中から (おそらく別の) 接続を検出して使用することになります。

MQSeries-Bridge キューを論理的に支えている MQSeries 接続が解放されて再割り振りされるということは、キューのユーザーからは意識されません。ただし、アイドル状態の接続が接続プールに入ったり、接続プールから出たりするときに、わずかながら時間がかかるということはあるでしょう。

アイドル時間は、分単位で指定します。

タイマー・パラメーターは、 MQSeries-Bridge のハートビートのパラメーターの指定時間に直接影響されます。

「*MQSeries Everyplace for Multiplatforms プログラミング・ガイド*」の、MQSeries-Bridge オブジェクトの構成パラメーターを参照してください。

極端なケースとして、アイドル時間を 0 に指定した場合は、接続を使用し終えたらすぐに接続がプールに戻される、ということになります。確かにこの設定の場合は、ごく少数の MQSeries クライアント接続チャネルを非常に多くの MQSeries-Bridge キューで効率的に "共用" できますが (ただし、すべてのキューの MQSeries-Bridge/MQProxy/ClientConnection の詳細設定が同じということが前提)、メッセージが発生するたびに、プールへの解放とプールからの取得という処理が必要になってしまいます。

5 分というデフォルト値を推奨します。

MQeField ラベル: MQeMQBridgeQueueAdminMsg.Queue_MaxIdleTime

コンストラクター

MQeMQBridgeQueueAdminMsg

構文

```
1. public MQeMQBridgeQueueAdminMsg() throws Exception
2. public MQeMQBridgeQueueAdminMsg(String bridge,
                                   String mqQMgrProxy,
                                   String clientConnection
                                   int maxIdleTimeout) throws Exception
```

説明 2 つのコンストラクターがあります。

1. このバージョンは、 MQSeries-Bridge キューを管理するためのデフォルトの管理メッセージを作成して初期化します。
2. このバージョンは、MQSeries-Bridge の名前、プロキシーの名前、クライアント接続の名前のそれぞれの初期値を組み込みます。

パラメーター

bridge 管理メッセージの送信先である MQSeries-Bridge の名前を含むストリング。ヌルまたは "" に設定した場合は、何も設定しなかったことになります。

MQQMgrProxy MQSeries-Bridge キューの読み込み先として設定されている送信キューを所有する MQSeries キュー・マネージャーの名前を含むストリング。ヌル または "" に設定した場合は、何も設定しなかったことになります。

clientConnection

MQSeries キュー・マネージャーと通信するためのクライアント接続の名前を含むストリング。ヌル または "" に設定した場合は、何も設定しなかったことになります。

maxIdleTimeout

キューが、 MQSeries システムとの接続を未使用状態のまま保持しておける最大時間 (分単位)。キューが未使用のままこの時間を経過すると、接続は接続プールに戻されます。 0 を設定した場合は、接続は使用直後に接続プールに戻されます。

戻り値 なし

例外

java.lang.Exception メッセージを作成できなかった場合

例

1. ```
   MQeMQBridgeQueueAdminMsg msg;
   msg = new MQeMQBridgeQueueAdminMsg( );
   ```
2. ```
   MQeMQBridgeQueueAdminMsg msg;
   msg = new MQeMQBridgeQueueAdminMsg( "MQBridgeV100",
                                       "lizzieQM",
                                       "svrconn",
                                       5 );
   ```

メソッド

メソッド	目的
characteristics	キューの有効な特性を含むフィールド・オブジェクトを作成します。

MQeMQBridgeQueueAdminMsg characteristics

構文

```
public MQeFields characteristics() throws Exception
```

説明 キューの有効な特性を含むフィールド・オブジェクトを作成します。MQeFields オブジェクトは、有効なフィールド名を含んでいますが、各特性の値は含んでいません。これは、すべての有効な特性の名前とタイプを判別するために使用できます。

MQeRemoteQueueAdminMsg クラスの **characteristics()** をオーバーライドします。

パラメーター

なし

戻り値 このキューの特性を含む MQeFields オブジェクト。

例外

java.lang.Exception	MQeFields オブジェクトが作成できない場合

例

```
MQeMQBridgeQueueAdminMsg msg;
msg = new MQeMQBridgeQueueAdminMsg( "MQBridgeV100",
                                    "lizzieQM",
                                    "svrconn",
                                    5 );
MQeFields adminMsgChars = msg.characteristics();
```

MQeMQBridges

MQeMQBridges クラスは、すべての MQSeries-Bridge の ″ローダー″ の役目を果たします。これは、 ini ファイル内で MQeMQBridge クラス別名として指定されるクラスであり、このクラスのインスタンスが MQSeries Everyplace サーバーによって動的にロードされるようにします。

MQeMQBridges クラスの目的は、レジストリーからメモリーに MQSeries-Bridge オブジェクトをロードすることと、 JVM 内の使用可能な MQSeries-Bridge のリストを保守することです。1 つの JVM には、 MQeMQBridge オブジェクトは 1 つしかありません。

サーバー・コードは、 **constructor()** の直後に **activate()** メソッドを呼び出し、すべての MQSeries-Bridge をクリーンにシャットダウンするときに、 **close()** メソッドを呼び出します。 MQeMQBridges オブジェクトを開始してからシャットダウンするまで、サーバーは、 MQeMQBridges オブジェクトでガーベッジ・コレクションが行われるのを防ぐために、そのオブジェクトへの参照を保持していなければなりません。

パッケージ **com.ibm.mqe.mqbridge**

このクラスは、 **MQeAdministeredObject** の拡張クラスです。

コンストラクターの要約

MQeMQBridges

構文

```
public MQeMQBridges() throws Exception
```

説明 シンプルなコンストラクター。

ゲートウェイは、MQSeries Everyplace クラス・ローダーを使ってこのクラスをロードするはずです。したがって、このコンストラクターが直接呼び出されることはありません。それからサーバーは、任意の構成パラメーターを渡して **activate()** メソッドを呼び出します。

パラメーター

なし

戻り値 なし

例外 java.lang.Exception

例

```
MQeMQBridges mqBridges = (MQeMQBridges) MQe.loader.loadObject( "MQBridge" );
```

メソッドの要約

メソッド	目的
activate	MQeMQBridges オブジェクトを活動化するために、コンストラクターの直後に、ゲートウェイによって呼び出されます。
close	MQeMQBridge オブジェクトをクローズします。

MQeMQBridges activate

構文

```
public void activate(com.ibm.mqe.MQeFields config) throws Exception
```

説明

ヌル・コンストラクターの直後に、MQSeries Everyplace サーバーによって呼び出されます。これを使用して、構成を MQeMQBridges オブジェクトに渡し、このオブジェクトが構成の中で指定されている任意の MQSeries-Bridge をロードできるようにします。

パラメーター

config MQSeries Everyplace サーバーの初期化に使用された ini ファイルの完全な内容を含む MQeFields オブジェクト。

```
Fields=Aliases
        ...
        (ascii)MQBridge=com.ibm.mqe.mqbridge.MQeMQBridges
    Fields=ChannelManager
       ...
    Fields=Listener
        ...
    Fields=QueueManager
        ...
    Fields=MQBridge
    (ascii)LoadBridgeRule=RuleClass
```

戻り値 なし

例外

java.lang.Exception 基礎となるサーバーが MQeMQBridges オブジェクトを活動化できなかった場合、または指定した構成パラメーターが無効だった場合は失敗

例

```
mqBridges.activate( myConfigFields );
```

MQeMQBridges close

構文

```
public void close() throws Exception
```

説明 MQeMQBridges オブジェクトをできるだけクリーンに閉じます。

パラメーター

なし

戻り値 なし

例外

java.lang.Exception	オブジェクトをクリーンにシャットダウンできなかった場合

例

```
mqBridges.close( );
```

MQeMQBridgesAdminMsg

MQeMQBridges オブジェクトに対して実行する管理コマンドをカプセル化するために使用します。

このメッセージは、管理を担当しているアプリケーションによって生成されます。

このクラスには、ターゲット MQSeries Everyplace システムで実行されるロジックも入っています。

管理キューは、performAction メソッドを呼び出します。

パッケージ **com.ibm.mqe.mqbridge**

このクラスは、 MQeAdminMsg の拡張クラスです。

定数と変数

MQeQueueAdminMsg には、 MQeAdminMsg の定数と変数のほかに、以下の定数と変数が用意されています。

Action_Start
管理対象オブジェクトを開始するための操作コード。

```
public static final int Action_Start
```

Action_Stop
管理対象オブジェクトを停止するための操作コード。

```
public static final int Action_Stop
```

コンストラクター

MQeMQBridgesAdminMsg

構文

1. `public MQeMQBridgesAdminMsg() throws Exception`
2. `public MQeMQBridgesAdminMsg(boolean affectChildren) throws Exception`

説明 2 つのコンストラクターがあります。

1. このバージョンは、デフォルトの MQeMQBridgesAdminMsg を作成して初期化します。
2. このバージョンは、管理コマンドによって子オブジェクトが影響を受けるかどうかを指定するフラグを組み込みます。

パラメーター

affectChildren この管理メッセージが、 MQeMQBridges オブジェクトの子オブジェクトに影響を及ぼすかどうかを設定するためのブール・フラグ。

true は、子オブジェクトに影響が及ぶことを意味し、 **false** は、子オブジェクトに影響が及ばないことを示します。

一部のコマンド (停止コマンドなど) は、どんな場合にも子オブジェクトに影響を及ぼします。

このフラグは、ブリッジの管理対象オブジェクトに送られる MQeEvent に転送されることもあります。

戻り値 なし

例外

java.lang.Exception	`affectChildren` フラグを保持するフィールドを、作成できなかった場合

例

```
MQeMQBridgesAdminMsg msg = new MQeMQBridgesAdminMsg(true);
```

関連する関数

MQeEvent

メソッド

メソッド	目的
characteristics	このタイプの管理メッセージで必要なすべてのフィールドを含む MQeFields オブジェクトを作成します。
getName	作成される管理対象オブジェクトの名前を取得します。
start	管理対象オブジェクトを開始するよう指定します。
stop	管理対象オブジェクトを停止するよう指定します。

MQeMQBridgesAdminMsg characteristics

構文

```
public MQeFields characteristics() throws Exception
```

説明 このタイプの管理メッセージで必要なすべてのフィールドを含む MQeFields オブジェクトを作成します。

MQeBridgesAdminMsg クラスの **characteristics()** をオーバーライドします。

パラメーター

なし

戻り値 このリソースの特性を含む MQeFields オブジェクト。

例外

java.lang.Exception	MQeFields オブジェクトを作成できなかった場合

例

```
MQeMQBridgesAdminMsg msg = new MQeMQBridgesAdminMsg(true);
MQeFields theseCharacteristics = msg.characteristics();
```

MQeMQBridgesAdminMsg getName

構文

```
public String getName()
```

説明

作成する管理対象オブジェクトの名前を戻します。

MQeAdminMsg クラスの **getName()** をオーバーライドします。

パラメーター

なし

戻り値 名前が設定されていない場合はヌル、そうでない場合はストリング。

例外 なし

例

```
MQeMQBridgesAdminMsg msg = new MQeMQBridgesAdminMsg(true);
String name = msg.getName();
```

MQeMQBridgesAdminMsg start

構文

1. `public void start() throws Exception`
2. `public void start(boolean affectChildren) throws Exception`
3. `public void start(MQeFields fields) throws Exception`

説明 3 つのバージョンがあります。

1. このバージョンは、管理メッセージの送信元によって使用され、管理対象オブジェクトを開始するように指定します。さらに、そのすべての子オブジェクトも開始するように指定します。
2. このバージョンは、管理メッセージの送信元によって使用され、管理対象オブジェクトを開始するように指定します。パラメーターの値によっては、さらに、そのすべての子オブジェクトも開始するように指定します。
3. このバージョンは、管理対象オブジェクトを開始するように指定し、さらに、*affectChildren* フィールドの値に基づいて、そのすべての子オブジェクトを開始するかどうかを指定します。

   その他の MQSeries-Bridge 関連パラメーターも MQeFields オブジェクトに渡すことができます。

パラメーター

affectChildren この管理メッセージに設定されているコマンド操作が、

MQeMQBridges オブジェクトの子オブジェクトに影響を及ぼすかどうかを設定するためのブール。

true は子オブジェクトに影響が及ぶことを意味し、 **false** は子オブジェクトに影響が及ばないことを示します。

一部のコマンド (停止コマンドなど) は、どんな場合にも子オブジェクトに影響を及ぼします。

このフラグは、 MQeMQBridges 管理対象オブジェクトに送られる **MQeEvent** に、転送されることもあります。

fields MQSeries-Bridge の名前、*affectChildren* フラグ、その他の MQSeries-Bridge 関連パラメーターのフィールド・セットを含む MQeFields オブジェクト。

これらのフィールドは、送信準備のできたこの管理メッセージにそのままコピーされます。フィールドのパラメーターに関する妥当性検査は行われません。

affectChildren フラグがある場合は、 **true** のデフォルト値がオーバーライドされます。

戻り値 なし

例外

java.lang.Exception 開始アクション、**affectChildren** フラグ、または渡されたフィールドのいずれかを、この管理メッセージに設定できない場合

例

1. ```
   MQeMQBridgesAdminMsg msg = new MQeMQBridgesAdminMsg();
   msg.start();
   ```
2. ```
   MQeMQBridgesAdminMsg msg = new MQeMQBridgesAdminMsg();
   msg.start(true);
   ```
3. ```
   MQeMQBridgesAdminMsg msg = new MQeMQBridgesAdminMsg();
   MQeFields fields = new MQeFields();
   fields.putBoolean( MQeCharacteristicLabels.MQE_FIELD_LABEL_AFFECT_CHILDREN,
                      true );
   msg.start(fields);
   ```

MQeMQBridgesAdminMsg stop

構文

1. ```
   public void stop() throws Exception
   ```
2. ```
   public void stop(MQeFields fields) throws Exception
   ```

説明 2 つのバージョンがあります。

1.
   ```

このバージョンは、管理メッセージの送信元によって使用され、管理対象オブジェクトを停止するように指定します。さらに、そのすべての子オブジェクトも停止するように指定します。

2.

このバージョンは、管理対象オブジェクトを停止するように指定し、さらに、そのすべての子オブジェクトも停止するように指定します。

このバージョンは、どの MQeAdministeredObject を停止するかを識別するフィールドを含むフィールド・セットを受け入れます。

パラメーター

fields	MQeFields オブジェクト。これらのフィールドは、管理メッセージのフィールドにコピーされ、現在のフィールド値をオーバーライドします。

戻り値 なし

例外

java.lang.Exception	この管理メッセージにアクションを設定できなかった場合

例

1.
```
MQeMQBridgesAdminMsg msg = new MQeMQBridgesAdminMsg();
msg.start();
```

2.
```
MQeMQBridgesAdminMsg msg = new MQeMQBridgesAdminMsg();
MQeFields fields = new MQeFields();
fields.putBoolean( MQeCharacteristicLabels.MQE_FIELD_LABEL_AFFECT_CHILDREN,
                   true );
msg.start(fields);
```

MQeMQQMgrProxyAdminMsg

このメッセージは、管理を担当しているアプリケーションによって作成され、MQeMQQMgrProxy オブジェクトに対して実行される管理コマンドをカプセル化するために使用されます。

このクラスには、ターゲット MQSeries Everyplace システムで実行されるロジックも入っています。

管理キューは、performAction メソッドを呼び出します。

パッケージ **com.ibm.mqe.mqbridge**

このクラスは、 MQeMQBridgeAdminMsg の拡張クラスです。

コンストラクター

MQeMQQMgrProxyAdminMsg

構文

1. `public MQeMQQMgrProxyAdminMsg() throws Exception`
2.
```
public MQeMQQMgrProxyAdminMsg(String bridge,
                              String MQQMgrProxy,
                              boolean affectChildren) throws Exception
```

説明 2 つのコンストラクターがあります。

1. このバージョンは、デフォルトの MQeMQQMgrProxyAdminMsg を作成して初期化します。
2. このバージョンは、MQSeries Everyplace キュー・マネージャーの名前、MQSeries-Bridge の名前、プロキシーの名前を組み込みます。

パラメーター

bridge 管理メッセージの送信先である MQSeries-Bridge の名前を含むストリング。ヌル または "" に設定した場合は、何も設定しなかったことになります。

MQQMgrProxy 管理メッセージの送信先であるプロキシーの名前を含むストリング。ヌル または "" に設定した場合は、何も設定しなかったことになります。

affectChildren この管理メッセージが、 MQeMQBridges オブジェクトの子オブジェクトに影響を及ぼすかどうかを設定するためのブール・フラグ。

true は子オブジェクトに影響が及ぶことを意味し、 **false** は子オブジェクトに影響が及ばないことを示します。

戻り値 なし

例外

java.lang.Exception	どのフィールドも設定できなかった場合、または MQSeries キュー・マネージャー名が無効である場合

例

```
1. MQeMQQMgrProxyAdminMsg msg = new MQeMQQMgrProxyAdminMsg();
2. MQeMQQMgrProxyAdminMsg msg;
   msg = new MQeMQQMgrProxyAdminMsg("MQBridgeV100", "lizzieQM");
```

メソッド

メソッド	目的
characteristics	このタイプの管理メッセージで必要なすべてのフィールドを含む、 MQSeries Everyplace フィールド・オブジェクトを作成します。
getMQQMgrProxyName	管理対象オブジェクトから、 MQSeries キュー・マネージャー・プロキシー名を取得します。
getName	作成する管理対象オブジェクトの名前を取得します。
putMQQMgrProxyName	MQeFields 管理メッセージ・オブジェクト内のフィールドに、MQSeries キュー・マネージャー・プロキシー名を入れます。
setName	MQeFields 管理メッセージ・オブジェクト内のフィールドに、名前情報を入れます。

MQeMQQMgrProxyAdminMsg characteristics

構文

```
public MQeFields characteristics() throws Exception
```

説明 このタイプの管理メッセージで必要なすべてのフィールドを含む、 MQeFields オブジェクトを作成します。

このリソースの特性を含むフィールド・オブジェクトを戻します。生成されるフィールド・オブジェクトから、このリソースのフィールド名とフィールド・タイプの完全セットが判別できます。

MQeBridgeAdminMsg クラスの **characteristics()** をオーバーライドします。

パラメーター

なし

戻り値 このリソースの特性を含む MQeFields オブジェクト。

例外

java.lang.Exception	MQeFields オブジェクトを作成できなかった場合

例

```
MQeMQQMgrProxyAdminMsg msg;
msg = new MQeMQQMgrProxyAdminMsg("MQBridgeV100", "lizzieQM");
MQeFields proxyChars = msg.characteristics();
```

MQeMQQMgrProxyAdminMsg getMQQMgrProxyName

構文

```
public String getMQQMgrProxyName() throws Exception
```

説明

管理対象オブジェクトから、 MQSeries キュー・マネージャー・プロキシー名を取得します。

このメソッドは、フィールドの妥当性も検査します。
MQeMQQMgrProxyAdminMsg またはその下位オブジェクトのいずれかに対して実行できます。

パラメーター
なし

戻り値 なし

例外

java.lang.Exception	取得が失敗した場合

例

```
MQeMQQMgrProxyAdminMsg msg;
msg = new MQeMQQMgrProxyAdminMsg("MQBridgeV100", "lizzieQM");
String proxyName = msg.getMQQMgrProxyName();
```

MQeMQQMgrProxyAdminMsg getName

構文

```
public String getName()
```

説明

現在の管理対象オブジェクトの名前を戻します。このクラスのオブジェクトに対して実行される場合は、 `getMQQMgrProxyName()` と同じ結果になります。

MQeMQBridgeAdminMsg クラスの **getName()** をオーバーライドします。

パラメーター
なし

戻り値 名前が設定されていない場合はヌル、それ以外の場合はストリング。

例外 なし

例

```
MQeMQQMgrProxyAdminMsg msg;
msg = new MQeMQQMgrProxyAdminMsg("MQBridgeV100", "lizzieQM");
String proxyName = msg.getMName();
```

MQeMQQMgrProxyAdminMsg putMQQMgrProxyName

構文

```
public void putMQQMgrProxyName(String mqQMgrProxy) throws Exception
```

説明

MQeFields 管理メッセージ・オブジェクト内のフィールドに、 MQSeries キュー・マネージャー・プロキシー名を入れます。

パラメーター

mqQMgrProxy 管理メッセージの送信先であるプロキシーの名前を含むストリング。ヌル または "" に設定した場合は、何も設定しなかったことになります。

戻り値 なし

例外

java.lang.Exception どのフィールドも設定できなかった場合、または MQSeries キュー・マネージャー名が無効である場合

例

```
MQeMQQMgrProxyAdminMsg msg = new MQeMQQMgrProxyAdminMsg();
msg.putMQQMgrProxyName("lizzieQM");
```

MQeMQQMgrProxyAdminMsg setName

構文

```
public void setName(String bridge,
                    String mqQMgrProxy) throws Exception
```

説明

管理メッセージ・オブジェクト内に、MQSeries-Bridge の名前とプロキシー名を入れます。

パラメーター

bridge 管理メッセージの送信先である MQSeries-Bridge の名前を含むストリング。ヌル または "" に設定した場合は、何も設定しなかったことになります。

mqQMgrProxy 管理メッセージの送信先であるプロキシーの名前を含むストリング。ヌル または "" に設定した場合は、何も設定しなかったことになります。

戻り値 なし

例外

java.lang.Exception	どのフィールドも設定できなかった場合、または MQSeries キュー・マネージャー名が無効である場合

例

```
MQeMQQMgrProxyAdminMsg msg = new MQeMQQMgrProxyAdminMsg();
msg.setName("MQBridgeV100", "lizzieQM");
```

MQeRunState

管理対象オブジェクトの ″実行状態″ (実行、停止、静止、開始など) を保持するクラス。

パッケージ **com.ibm.mqe.mqbridge**

このクラスは、 **MQeBridgeServices** の拡張クラスです。

定数と変数

MQeQueueAdminMsg には、MQeBridgeServices の定数と変数のほかに、以下の定数と変数が用意されています。

RUN_STATE_RUNNING

管理対象オブジェクトがアクティブな場合の状態。

```
public static final int RUN_STATE_RUNNING
```

RUN_STATE_STOPPED

管理対象オブジェクトが非アクティブな場合の状態。

```
public static final int RUN_STATE_STOPPED
```

MQeTransformerInterface

MQSeries メッセージを MQeMsgObjects に (またはその逆に) 変換するすべてのクラスは、このインターフェースに準拠しなければなりません。

パッケージ **com.ibm.mqe.mqbridge**

メソッド

メソッド	目的
activate	このインターフェースをインプリメントするクラスに対して、トランスフォーマー定義で指定されている可能性があるパラメーターを通知します。
transform	指定の MQSeries メッセージを MQeMsgObject に (またはその逆に) 変換します。

MQeTransformerInterface activate

構文

```
public abstract void activate(StringTokenizer params) throws Exception
```

説明 このインターフェースをインプリメントするクラスに対して、トランスフォーマー定義で指定されている可能性があるパラメーターを通知します。

パラメーター

params トランスフォーマー定義パラメーターを含む StringTokenizer。

戻り値 なし

例外

java.lang.Exception パラメーターが間違っていたり無効だったりした場合、またはトランスフォーマーを初期化するときに問題が発生した場合

MQeTransformerInterface transform

構文

1.
```
public abstract MQeMsgObject transform(MQMessage msg,
                                       String remoteQMgrName,
                                       String remoteQName) throws Exception
```

2.
```
public abstract MQMessage transform(MQeMsgObject msg,
                                    MQeMQBridgeQueue queue,
                                    MQPutMessageOptions pmo) throws Exception
```

説明 このメソッドには 2 つのバージョンがあります。

1. このバージョンは、指定の MQSeries メッセージを MQeMessageObject に変換します。
2. このバージョンは、指定の MQeMsgObject を MQSeries メッセージに変換します。

パラメーター

msg 変換される MQSeries または MQSeries Everyplace メッセージ。

remoteQMgrName
宛先 MQSeries Everyplace キュー・マネージャーの名前 (MQSeries 上のリモート・キュー定義から取得)。

remoteQName 宛先 MQSeries Everyplace キューの名前 (MQSeries 上のリモート・キュー定義から取得)。

msg 変換される MQSeries Everyplace メッセージ。

queue メッセージを受け入れたブリッジ・キューへの参照。

remoteQName 宛先 MQSeries Everyplace キューの名前 (MQSeries 上のリモート・キュー定義から取得)。

pmo トランスフォーマーによって修正できるブランク MQPutMessageOptions オブジェクトへの参照。ユーザーはこのパラメーターを使って、新しい MQSeries Everyplace メッセージを MQSeries に書き込むために必要な、任意のコンテキスト・オプションを指定できます。

戻り値

1. MQeMsgObject 形式の変換済みメッセージ。
2. MQSeries メッセージ形式の変換済みメッセージ。

例外

java.lang.Exception いずれかのパラメーターが無効である場合、メッセージのフォーマットがこのトランスフォーマーによって認識されていない場合、または変換中に問題が発生した場合

第9章 com.ibm.mqe.adapters のクラス

この節には、以下の MQSeries Everyplace クラスについての詳細情報が載せられています。

表 19. com.ibm.mqe.adapters のクラス群

クラス名	目的
MQeDiskFieldsAdapter	ローカル・ディスクに対する MQeFields 情報の読み取りと書き込みをサポートします。
MQeMemoryFieldsAdapter	MQeFields 情報を一時的に保管します。
MQeReducedDiskFieldsAdapter	ローカル・ディスクに対する MQeFields 情報の高速書き込みをサポートします。
MQeTcpipAdapter	TCP/IP ストリームにおけるデータの読み取りと書き込みをサポートします。
MQeTcpipHttpAdapter	MQeTcpipAdapter の拡張クラスであり、 HTTP 1.0 プロトコルの基本的なサポートを提供します。
MQeTcpipLengthAdapter	MQeTcpipAdapter の拡張クラスであり、バイト効率のよいシンプルなプロトコルを提供します。
MQeTcpipHistoryAdapter	MQeTcpipLengthAdapter の拡張クラスであり、最近使用したデータをキャッシュに入れるための効率のよいプロトコルを提供します。
MQeUdpipAdapter	UDP/IP データグラムにおける保証されたデータ転送をサポートします。
MQeWESAuthenticationAdapter	Websphere Everyplace 認証プロキシーおよび透過プロキシーを介した、 HTTP 要求のトンネルをサポートします。

MQeDiskFieldsAdapter

MQeDiskFieldsAdapter は、ローカル・ディスクに対する MQeFields 情報の読み取りおよび書き込みをサポートするアダプターです。このクラスでは、1 つ 1 つのインスタンスがディスク上の 1 つのファイルを表しています。

パッケージ **com.ibm.mqe.adapter**

このクラスは、**MQeAdapter** の下位クラスです。

メソッド

メソッド	目的
accept	指定されたストリングの拡張子を調べ、現在定義されているファイル・フィルターであるかどうかを確認します。
activate	新規の MQeDiskFieldsAdapter オブジェクトを初期設定します。
close	現在の MQeDiskFieldsAdapter オブジェクトをクローズします。
control	ファイル・フィルターの設定や、現行ファイルのフォルダーに含まれているファイルのリスト表示を可能にします。
erase	ディスクからファイルを除去します。
open	使用するファイルをオープンします。
readObject	現在のファイルから MQeFields オブジェクトを読み取ります。
status	現在アクティブになっている設定についての情報を示します。
writeObject	ディスクに MQeFields オブジェクトを書き込みます。

MQeDiskFieldsAdapter accept

構文

```
public boolean accept( File dir, String name )
```

説明 指定されたストリングの拡張子を調べ、現在定義されているファイル・フィルターであるかどうかを確認します。

パラメーター

File dir 使用しない

String name 確認するファイルの名前

戻り値

True 指定されたパラメーターの末尾が現在のフィルター・ストリングである場合

例外 なし

例

```
Boolean isText = mydfa.accept(null, ".txt");
```

MQeDiskFieldsAdapter activate

構文

```
public void   activate( String  fileDesc,
                        Object  param,
                        Object  options,
                        int     lrecl,
                        int     noRec ) throws Exception
```

説明 このメソッドは新規の MQeDiskFieldsAdapter オブジェクトを初期設定します。ファイルあるいは MQeFields オブジェクトのサイズは、「サービス品質 (QOS) サイズ (Quality of Service (QOS) Size)」パラメーターで設定します。

パラメーター

String fileDesc オープンするディレクトリー・パス。ここでは、ファイルへのパスだけを指定します。ファイルをオープンしたいときは、そのファイルの名前を指定します。この値は、これ以降の数々のメソッドで、「*ファイル記述子 (File descriptor)*」として使用されます。

Object options 以下のオープン・オプションを指定するストリング

MQe.MQe_Adapter_READ
ファイルを読み取る

MQe.MQe_Adapter_WRITE
ファイルを書き込む

MQe.MQe_Adapter_UPDATE
ファイルの読み取りと書き込みの両方を行う

param 使用しない

lrecl 使用しない

norec 使用しない

戻り値 なし

例外

MQeException
指定されたディレクトリー階層を動的に作成できなかった場合

例

```
mydfa.activate("./mydir/mysubdir", null, MQe.MQeAdapter_READ, 0, 0);
```

MQeDiskFieldsAdapter close

構文

```
public void close( Object opt ) throws Exception
```

説明 現在のアダプターをクローズします。これにより、このアダプターに対して読み取りや書き込みを行うことはできなくなります。

パラメーター

なし

戻り値 なし

例外 なし

例

```
mydfa.close(null);
```

MQeDiskFieldsAdapter control

構文

```
public Object control( Object opt, Object ctrlObj ) throws Exception
```

説明 ファイル・フィルターの設定や、現在のディレクトリーに含まれているファイルのリスト表示を可能にします。

パラメーター

Object opt 次のいずれかの値が含まれているストリング

MQe_Adapter_FILTER
フィルターを 2 番目のパラメーターの値に設定する

MQe_Adapter_LIST
ファイルをリスト表示する

Object ctrlObj 使用するフィルターを指定するストリング (ただし、MQe_Adapter_FILTER が指定されている場合)。MQe_Adapter_LIST では使用されません。

戻り値 MQe_Adapter_FILTER が指定された場合は ヌル、 MQe_Adapter_LIST が指定された場合は String[] によるファイルのリスト

例外 なし

例

```
String[] fileList = (String[]) mydfa.control(MQe.MQe_Adapter_LIST,
                     null);
```

MQeDiskFieldsAdapter erase

構文

```
public void erase( Object opt ) throws Exception
```

説明 ディスクからファイルを削除します。ディレクトリーは削除しません。

パラメーター

Object opt 削除するファイル名を含むストリング。このファイル名は、オペレーティング・システムの削除ルーチンに渡される前に、「*ファイル記述子 (File descriptor)*」の現在の値に付加されます。

戻り値 なし

例外

MQeException
削除が失敗した場合、または指定された引き数がディレクトリーを参照していたり、ストリングではなかった場合

例

```
mydfa.erase("myfile.txt");
```

MQeDiskFieldsAdapter open

構文

```
public void open( Object opt ) throws Exception
```

説明 オープンしているファイルをすべてクローズし、使用するファイルをオープンします。

パラメーター

Object opt オープンするファイルの名前を指定するストリング。ここで指定できるファイルは、「*ファイル記述子 (File Descriptor)*」の値 (呼び出しの実行時に指定) で指定したフォルダーの中に存在しているファイルです。

戻り値 なし

例外 なし

例

```
mydfa.open("myfile.txt");
```

MQeDiskFieldsAdapter readObject

構文

```
public synchronized Object readObject( Object opt ) throws Exception {
```

説明 現在のファイルから MQeFields オブジェクトを読み取ります。

パラメーター

Object opt ディレクトリー・オブジェクト内でフィルターとして使用できる MQeFields オブジェクト。そのようなオブジェクトがない場合はヌル。

戻り値 読み取られた MQeFields オブジェクト

例外

MQeException
フィルターに一致するファイルが見つからない場合、または読み取るためのファイルがオープンされていない場合

例

```
MQeFields myFields = (MQeFields) mydfa.readObject(null);
```

MQeDiskFieldsAdapter status

構文

```
public String status( Object  opt  ) throws Exception
```

説明 現在の設定の詳細を示します。

パラメーター

Object opt 次のいずれかの値を含むストリング

MQe.MQe_Adapter_FILTER
現在アクティブになっているフィルターの名前を取得する

MQe.MQe_Adapter_FILENAME
現在アクティブになっているファイルの名前を取得する

戻り値 フィルター名またはファイル名のいずれかを含むストリング

例外 なし

例

```
String filename = (String) mydfa.status(MQe.MQe_Adapter_FILENAME);
```

MQeDiskFieldsAdapter writeObject

構文

```
public synchronized void writeObject(Object opt, Object data)
                    throws Exception
```

説明 ディスクに MQeFields オブジェクトを書き込みます。

パラメーター

Object opt 使用しない

Object data ディスクにダンプする MQeFields オブジェクト

戻り値 なし

例外

MQeException
無効なデータ引き数が指定された場合、または書き込むためのファイルがオープンされていなかった場合

例

```
mydfa.writeObject(null, myMQeFields);
```

MQeMemoryFieldsAdapter

MQeMemoryFieldsAdapter は、メモリーに対する MQeFields 情報の読み取りおよび書き込みをサポートするアダプターです。このアダプターは、ディレクトリー構造が実際は内部ハッシュ・テーブルであるという点を除いて、 MQeDiskFieldsAdapter と同じように動作します。

パッケージ **com.ibm.mqe.adapter**

このクラスは、MQeAdapter の下位クラスです。

メソッドの要約

メソッド	目的
activate	新規の MQeMemoryFieldsAdapter オブジェクトを初期設定します。
close	現在の MQeMemoryFieldsAdapter オブジェクトをクローズします。
control	現行ファイルのフォルダーに含まれているファイルのリスト表示を可能にします。
erase	メモリーからファイルを除去します。
open	使用するファイルをオープンします。
readObject	現在のファイルから MQeFields オブジェクトを読み取ります。
status	現在の設定についての情報を示します。
writeObject	メモリーに MQeFields オブジェクトを書き込みます。

MQeMemoryFieldsAdapter activate

構文

```
public void   activate( String  fileDesc,
                        Object  param,
                        Object  options,
                        int     lrecl,
                        int     noRec ) throws Exception
```

説明 このメソッドは新規の MQeMemoryFieldsAdapter オブジェクトを初期設定します。

パラメーター

String fileDesc オープンするディレクトリー・パス。ここでは、ファイルへのパスだけを指定します。ファイルをオープンしたいときは、そのファイルの名前を指定します。この値は、これ以降の数々のメソッドで、「*ファイル記述子 (File descriptor)*」として使用されます。

Object options 以下のオープン・オプションを指定するストリング

MQe.MQe_Adapter_READ
ファイルを読み取る

MQe.MQe_Adapter_WRITE
ファイルを書き込む

MQe.MQe_Adapter_UPDATE
ファイルの読み取りと書き込みの両方を行う

param 使用しない

lrecl 使用しない

norec 使用しない

戻り値 なし

例外 なし

例

```
mymfa.activate("./mydir/mysubdir", null,
               MQe.MQeAdapter_READ, 0, 0);
```

MQeMemoryFieldsAdapter close

構文

```
public void close( Object opt ) throws Exception
```

説明 現在のアダプターをクローズします。これにより、このアダプターに対して読み取りや書き込みを行うことはできなくなります。

パラメーター
なし

戻り値 なし

例外 なし

例

```
mymfa.close(null);
```

MQeMemoryFieldsAdapter control

構文

```
public Object control( Object opt, Object ctrlObj ) throws Exception
```

説明 ファイル・フィルターの設定や、現在のディレクトリーに含まれているファイルのリスト表示を可能にします。

パラメーター

Object opt 次のいずれかが入ったストリング・オブジェクト

MQe_Adapter_FILTER
フィルターを 2 番目のパラメーターの値に設定する

MQe_Adapter_LIST
ファイルをリスト表示する

Object ctrlObj 使用するフィルターを指定するストリング (ただし、MQe_Adapter_FILTER が指定されている場合)。それ以外の場合はヌル。

戻り値 MQe_Adapter_FILTER が指定された場合はヌル、 MQe_Adapter_LIST が指定された場合は String[] によるファイルのリスト

例外 なし

例

```
String]] fileList = (String]]) mydfa.control(MQe.MQe_Adapter_LIST,
                        null);
```

MQeMemoryFieldsAdapter erase

構文

```
public void erase( Object opt ) throws Exception
```

説明 メモリーからファイルを削除します。ディレクトリーは削除しません。

パラメーター

Object opt 削除するファイルの名前を含むストリング。このファイル名は、ハッシュ・テーブルの除去ルーチンに渡される前に、「*ファイル記述子 (File descriptor)*」の現在の値に付加されます。

戻り値 なし

例外

MQeException
指定された引き数がディレクトリーを参照していたり、ストリングではなかった場合

例

```
mydfa.erase("myfile.txt");
```

MQeMemoryFieldsAdapter open

構文

```
public void open( Object opt ) throws Exception
```

説明 オープンしているファイルをすべてクローズし、使用するファイルをオープンします。

パラメーター

Object opt オープンするファイルの名前を指定するストリング。ここで指定できるファイルは、「*ファイル記述子 (File Descriptor)*」の値 (呼び出しの実行時に指定) で指定したフォルダーの中に存在しているファイルです。

戻り値 なし

例外 なし

例

```
mymfa.open("myfile.txt");
```

MQeMemoryFieldsAdapter readObject

構文

```
public synchronized Object readObject( Object opt ) throws Exception
```

説明 現在のファイルから MQeFields オブジェクトを読み取ります。

パラメーター

Object opt ディレクトリー・オブジェクト内でフィルターの提供に使用できる MQeFields オブジェクト。そのようなオブジェクトがない場合はヌル。

戻り値 読み取られた MQeFields オブジェクト

例外

MQeException
フィルターに一致するファイルが見つからない場合、または読み取るためのファイルがオープンされていない場合

例

```
MQeFields myFields = (MQeFields) mymfa.readObject(null);
```

MQeMemoryFieldsAdapter status

構文

```
public String status( Object  opt  ) throws Exception
```

説明 現在の設定の詳細を示します。

パラメーター

Object opt 現在アクティブになっているファイルの名前を取得する MQe.MQe_Adapter_FILENAME が入ったストリング

戻り値 ファイルの名前が入ったストリング

例外 なし

例

```
String filename = (String) mymfa.status(MQe.MQe_Adapter_FILENAME);
```

MQeMemoryFieldsAdapter writeObject

構文

```
public synchronized void writeObject(Object opt, Object data)
                                        throws Exception
```

説明 メモリーに MQeFields オブジェクトを書き込みます。

パラメーター

Object opt 使用しない

Object data メモリーにダンプする MQeFields オブジェクト

戻り値 なし

例外

MQeException
無効なデータ引き数が指定された場合、または書き込むためのファイルがオープンしていなかった場合

例

```
mymfa.writeObject(null, myMQeFields);
```

MQeReducedDiskFieldsAdapter

MQeDiskFieldsAdapter は、ローカル・ディスクに対する MQeFields 情報の読み取りおよび書き込みをサポートするアダプターです。このアダプターは、MQeDiskFieldsAdapter の高性能バージョンです。

パッケージ **com.ibm.mqe.adapter**

このクラスは、 MQeDiskFieldsAdapter の拡張クラスです。

メソッド

メソッド	目的
writeObject	ディスクに MQeFields オブジェクトを書き込みます。

MQeDiskFieldsAdapter writeObject

構文

```
public synchronized void writeObject(Object opt, Object data)
                                              throws Exception
```

説明 ディスクに MQeFields オブジェクトを書き込みます。

パラメーター

Object opt 使用しない

Object data ディスクにダンプする MQeFields オブジェクト

戻り値 なし

例外

MQeException
無効なデータ引き数が指定された場合、または書き込むためのファイルがオープンしていなかった場合

例

```
myrdfa.writeObject(null, myMQeFields);
```

MQeTcpipAdapter

MQeTcpipAdapter は、 TCP/IP ストリームにおけるデータの読み取りおよび書き込みをサポートするアダプターです。

このアダプターには、データを交換するためのプロトコルが備わっていないため、このアダプターを使用して 2 つの MQSeries Everyplace アプリケーションを接続することはできません。ただし、このクラスを継承する種々のアダプターでは、いくつかのプロトコルが使用可能です。たとえば、MQeTcpipLengthAdapter や、 MQeTcpipHttpAdapter といったアダプターがあります。

このクラスのインスタンスは、着信する TCP/IP 接続の listen にも、既存のリスナーへの接続にも使用することができます。ただし、1 つの MQeTcpipAdapter オブジェクトでこの両方を行うことはできません。

パッケージ **com.ibm.mqe.adapter**

このクラスは、MQeAdapter の下位クラスです。

メソッドの要約

メソッド	目的
activate	新規の MQeTcpipAdapter オブジェクトを初期設定します。
close	現在の MQeTcpipAdapter オブジェクトをクローズします。
control	現在のアダプター・インスタンスに関する情報を設定または取得します。
open	使用するソケットをオープンします。
read	基礎となる TCP/IP ストリームからバイトを読み取ります。
readln	基礎となる TCP/IP ストリームからデータ行を読み取ります。
status	アダプターに関する情報を戻します。
write	基礎となる TCP/IP ストリームにバイトを書き込みます。
writeln	基礎となる TCP/IP ストリームにデータ行を書き込みます。

MQeTcpipAdapter activate

構文

```
public void   activate( String  fileDesc,
                        Object  param,
                        Object  options,
                        int     lrecl,
                        int     noRec ) throws Exception
```

説明 このメソッドは新規の MQeTcpipAdapter オブジェクトを初期設定します。MQeTcpipAdapter は、MQSeries Everyplace アプリケーションで使用するよう

には意図されておらず、このメソッドは、このクラスを拡張するすべてのアダプターの活動化メソッドの中で呼び出さなければなりません。

パラメーター

fileDesc このアダプターを表すストリング。
(たとえば ″Network:1.2.3.4:8082″)

param ACCEPT が出された場合に、作成されるアダプターに指定されるパラメーター・オブジェクト

options 以下のオープン・オプションを指定するストリングとして提供されるオブジェクト

MQe.MQe_Adapter_LISTEN
listen 用ソケットを作成する

MQe.MQe_Adapter_ACCEPT
接続を受け入れる

MQe.MQe_Adapter_LOCALHOST
ローカル・ホスト・アドレスを戻す

MQe.MQe_Adapter_NETWORK
ネットワーク・タイプ (TCPIP) を戻す

MQe.MQe_Adapter_PERSIST
呼び出しの受け入れ時に永続オブジェクトをコピーする

MQe.MQe_Adapter_QOSINPUTS
有効な「サービス品質 (QOS) (Quality of Service (QOS))」名を戻す

MQe.MQe_Adapter_SETSOCKET
予約済み

たとえば、listen 操作を実行するために TCP/IP ベースのアダプターを構成する場合は、このパラメーターに `MQe.MQe_Adapter_LISTEN` を指定します。

lrecl TCP/IP ブロック・サイズを指定する整数。ゼロより小さい値が指定された場合は、デフォルトの 4096 が使用されます。

norec ソケットのタイムアウト値 (ミリ秒) を指定する整数。

戻り値 なし

例外 なし

例

```
super.activate(fileDesc, param, options, lrecl,
        noRec);
```

MQeTcpipAdapter close

構文

```
public void close( Object opt ) throws Exception
```

説明 現在のアダプターをクローズします。これにより、このアダプターに対して読み取りや書き込みを行うことはできなくなります。

パラメーター

opt これがこのアダプターの最後のクローズである場合は、MQe.MQe_Adapter_FINAL を含むオブジェクト。MQe.MQe_Adapter_PERSIST オプションを設定してアダプターをオープンした場合は、クローズのメソッド呼び出しに MQe.MQe_Adapter_FINAL オプションを含めない限り、クローズは有効になりません。

戻り値 なし

例外 なし

例

```
super.close(null);
```

MQeTcpipAdapter control

構文

```
public Object control( Object opt, Object ctrlObj ) throws Exception
```

説明 現在のアダプター・インスタンスに関する情報を設定または取得します。

パラメーター

opt 以下のいずれかを含むストリングとして指定されるオブジェクト

MQe.MQe_Adapter_ACCEPT
接続を待機し、受け入れる (listen ソケット)

MQe.MQe_Adapter_SETSOCKET
提供されたソケット・オブジェクトを使用する

MQe.MQe_Adapter_PULSE
タイマー・パルス・スレッドを開始する

MQe.MQe_Adapter_QOSINPUTS -
「サービス品質 (QOS) (Quality of Service (QOS))」入力パラメーターのリストを戻す

;

ctrlObj MQe.MQe_Adapter_ACCEPT が指定された場合は、接続アダプタ

ー (たとえば ″HTTP:1.2.3.4...″) の「*ファイル記述子 (file descriptor)*」を表すストリング。 MQe.MQe_Adapter_SETSOCKET が指定された場合は、使用されるソケット。

戻り値 MQe.MQe_Adapter_ACCEPT が指定された場合は、接続要求によって作成されたアダプター・オブジェクト。それ以外の場合は、ヌル、または要求された情報を包含するオブジェクト。

例外

MQeException
指定されたオプションに対してパラメーター・オブジェクトが適切でなかった場合

例

```
super.control(opt, ctrlObj);
```

MQeTcpipAdapter open

構文

```
public void open( Object opt ) throws Exception
```

説明 使用するソケットをオープンします。これが listen アダプターである場合は、新しいサーバー・ソケットが作成されます。それ以外の場合は新しいソケットが作成されます。

パラメーター

opt 使用しない

戻り値 なし

例外 なし

例

```
super.open(null);
```

MQeTcpipAdapter read

構文

```
public byte[] read( Object opt, int recordSize )
                    throws Exception
```

説明 基礎となる TCP/IP ストリームからバイトを読み取ります。

パラメーター

opt 使用しない

recordSize 1 回の読み取り操作で読み取るバッファー・サイズの最大値を指定する整数。このパラメーターがゼロ以下である場合

は、デフォルトとして、「サービス品質 (QOS) (Quality of Service (QOS))」オブジェクトの「サイズ (Size)」パラメーターが使用されます。

戻り値 指定されたバイト数を最大値としたバイト配列のデータ。ブロックは、要求されたバイト数よりも小さい配列になる場合があります。

例外

EOFException
データがこれ以上使用できない場合

例

```
byte[] results = super.read(null, 4096);
```

MQeTcpipAdapter readln

構文

```
public String readln( Object opt ) throws Exception
```

説明 基礎となる TCP/IP ストリームからテキスト行を読み取ります。

パラメーター

opt 使用しない

戻り値 読み取られた行を含むストリング

例外

EOFException
データがこれ以上使用できない場合

例

```
String results = super.readln(null);
```

MQeTcpipAdapter status

構文

```
public String status( Object  opt  ) throws Exception
```

説明 現在の設定の詳細を示します。

パラメーター

opt 次のいずれかを含むストリング

MQe.MQe_Adapter_NETWORK
使用中のネットワークのタイプを取得する

MQe.MQe_Adapter_LOCALHOST
ローカル・ホスト名を取得する

戻り値 MQe.MQe_Adapter_NETWORK の場合はリテラル 'TCPIP'、
MQe.MQe_Adapter_LOCALHOST の場合はローカル・マシン名

例外 なし

例

```
String netType = (String) super.status(MQe.MQe_Adapter_NETWORK);
```

MQeTcpipAdapter write

構文

```
public void write( Object opt, int recordSize, byte data[] )
                   throws Exception
```

説明 指定されたバイト配列を TCP/IP 出力ストリームに書き込みます。

パラメーター

opt 書き込みに使用するオプションを含むストリング。現在では、MQe.MQe_Adapter_FLUSH しかサポートされていません。このオプションは、現在バッファーに入っているデータをフラッシュします。

recordSize 書き込むバイト数を指定する整数

byte[] data 書き込まれるデータ

戻り値 なし

例外 なし

例

```
super.write(MQe.MQe_Adapter_FLUSH, 4096, myByteArray);
```

MQeTcpipAdapter writeln

構文

```
public void writeln( Object opt, String data ) throws Exception
```

説明 指定されたストリングを出力ストリームに書き込み、改行します。

パラメーター

opt 書き込みに使用するオプションを含むストリング。現在では、MQe.MQe_Adapter_FLUSH しかサポートされていません。このオプションは、現在バッファーに入っているデータをフラッシュします。

data 書き込まれるデータを含むストリング。

戻り値 なし

例外 なし

例

```
super.writeln(null, "Hello");
```

MQeTcpipHttpAdapter

MQeTcpipHttpAdapter は、 MQeTcpipAdapter に HTTP 1.0 プロトコルのサポートを加えます。これによって、2 つの MQSeries Everyplace システムが TCP/IP ネットワーク上で通信を行うことが可能になります。

このアダプターによって、 TCP/IP ストリーム上のデータ・フローに HTTP ヘッダーを付加する読み取りや書き込みを、呼び出すことができます。通常、HTTP フローはファイアウォール上の特定のポートを通して許可されるため、このアダプターを使用すれば、MQSeries Everyplace データにファイアウォールを通過させ、 World Wide Web にアクセスすることができます。

パッケージ **com.ibm.mqe.adapter**

このクラスは、MQeTcpipAdapter の下位クラスです。ここにリスト表示されていない通常のアダプター操作すべては、このスーパークラスで提供される実装を使用します。

メソッド

メソッド	目的
read	基礎となる TCP/IP ストリームからバイトを読み取ります。
readEOF	ファイル終わりのマーカーまで、基礎となる TCP/IP ストリームからバイトを読み取ります。
readln	基礎となる TCP/IP ストリームからデータ行を読み取ります。
write	基礎となる TCP/IP ストリームにバイトを書き込みます。
writeln	基礎となる TCP/IP ストリームにデータ行を書き込みます。

MQeTcpipHttpAdapter read

構文

```
public byte[] read( Object opt, int recordSize )
                    throws Exception
```

説明 基礎となる TCP/IP ストリームからバイトを読み取ります。

パラメーター

opt 以下のいずれかになります。

MQe.MQe_Adapter_HEADER
HTTP ヘッダー情報を読み取る

MQe.MQe_Adapter_CONTENT
データ内容を読み取る

int recordSize 1 回の読み取り操作で読み取るバッファー・サイズの最大値

戻り値 指定されたバイト数を最大値としたバイト配列のデータ。ブロックは、要求されたバイト数よりも小さい配列になる場合があります。

例外

EOFException
データがこれ以上使用できない場合

例

```
byte[] results = myHttpAdapter.read
        (MQe.MQe_Adapter_HEADER + MQe.MQe_Adapter_CONTENT,
        4096);
```

MQeTcpipHttpAdapter readEOF

構文

```
public byte[] readEOF( Object opt ) throws Exception
```

説明 ファイル終わりのマーカーまで、基礎となる TCP/IP ストリームからバイトを読み取ります。 `MQeTcpipHttpAdapter.read(opt, -1)` と同じ意味を持ちます。

パラメーター

opt 以下のいずれかになります。

MQe.MQe_Adapter_HEADER
HTTP ヘッダー情報を読み取る

MQe.MQe_Adapter_CONTENT
データ内容を読み取る

戻り値 指定されたバイト数を最大値としたバイト配列のデータ。ブロックは、要求されたバイト数よりも小さい配列になる場合があります。

例外

EOFException
データがこれ以上使用できない場合

例

```
byte[] results = myHttpAdapter.readEOF(MQe.MQe_Adapter_HEADER);
```

MQeTcpipHttpAdapter readln

構文

```
public byte[] readln( Object opt ) throws Exception {
```

説明 基礎となる TCP/IP ストリームから行を読み取ります。

パラメーター

opt 以下のいずれかになります。

MQe.MQe_Adapter_HEADER
HTTP ヘッダー情報を戻す

NULL. 入力ストリームから次の行を戻す

戻り値 読み取られた行を含むストリング

例外

EOFException
データがこれ以上使用できない場合

例

```
String top = myHttpAdapter.readln(MQe.MQe_Adapter_HEADER);
```

MQeTcpipHttpAdapter write

構文

```
public void write( Object opt, int recordSize, byte data[] )
                   throws Exception
```

説明 指定されたバイト配列を TCP/IP 出力ストリームに書き込みます。なお、送信側に不必要なデータを残さないため、書き込みの終わったデータ・ストリームはフラッシュされます。

パラメーター

opt 以下のいずれかになります。

MQe.MQe_Adapter_HEADER
ヘッダー情報を付加する (POST 要求として)

MQe.MQe_Adapter_HEADERSP
データに HTTP 応答を付加する (200 OK)

recordSize 書き込むバイト数を指定する整数

data 書き込まれるデータ

戻り値 なし

例外 なし

例

```
myHttpAdapter.write(MQe.MQe_Adapter_HEADER, 4096,
                    myByteArray);
```

MQeTcpipHttpAdapter writeln

構文

```
public void writeln( Object opt, String data ) throws Exception
```

説明 指定されたストリングを出力ストリームに書き込みます。なお、送信側に不必要なデータを残さないため、書き込みの終わったデータ・ストリームはフラッシュされます。

パラメーター

opt 以下のいずれかになります。

MQe.MQe_Adapter_HEADER
ヘッダー情報を付加する (GET 要求として)

MQe.MQe_Adapter_HEADERSP
データに HTTP 応答を付加する (200 OK)

data 書き込まれるデータ

戻り値 なし

例外 なし

例

```
myHttpAdapter.writeln(MQe.MQe_Adapter_HEADER,
                      "Hello");
```

MQeTcpipLengthAdapter

MQeTcpipLengthAdapter は、 MQeTcpipAdapter に簡単なプロトコルを追加することによって、 2 つの MQSeries Everyplace システムが TCP/IP ネットワーク上で通信を行うことを可能にします。

このアダプターは、 MQeTcpipAdapter からの読み取りまたは書き込みの呼び出しをオーバーライドして、各データ・フローに 1 ～ 3 バイトのヘッダーを付加します。このヘッダーには、それぞれのデータの長さが示されます。このプロトコルを使用して、1,073,741,823 バイト (230 - 1) までのフローが可能です。

パッケージ **com.ibm.mqe.adapter**

このクラスは、MQeTcpipAdapter の下位クラスです。ここにリスト表示されていない通常のアダプター操作はすべて、このスーパークラスで提供される実装を使用します。

メソッド

メソッド	目的
activate	新規の MQeTcpipLengthAdapter オブジェクトを初期設定します。
read	基礎となる TCP/IP ストリームからバイトを読み取ります。
write	基礎となる TCP/IP ストリームにバイトを書き込みます。

MQeTcpipLengthAdapter activate

構文

```
public void   activate( String  fileDesc,
                        Object  param,
                        Object  options,
                        int     lrecl,
                        int     noRec ) throws Exception
```

説明 このメソッドは新規の MQeTcpipLengthAdapter オブジェクトを初期設定します。

パラメーター

fileDesc このアダプターを表すストリング
(たとえば ″Network:1.2.3.4:8082″)

param ACCEPT が出された場合に、作成されるアダプターに指定されるパラメーター

options 以下のオープン・オプションを指定するストリングとして提供されるオブジェクト

MQe.MQe_Adapter_LISTEN
listen 用ソケットを作成する

MQe.MQe_Adapter_ACCEPT
接続を受け入れる

MQe.MQe_Adapter_LOCALHOST
ローカル・ホスト・アドレスを戻す

MQe.MQe_Adapter_NETWORK
ネットワーク・タイプ (TCPIP) を戻す

MQe.MQe_Adapter_PERSIST
呼び出しの受け入れ時に永続オブジェクトをコピーする

MQe.MQe_Adapter_QOSINPUTS
有効な「サービス品質 (QOS) (Quality of Service (QOS))」名を戻す

MQe.MQe_Adapter_SETSOCKET
予約済み

lrecl TCP/IP ブロック・サイズを指定する整数。ゼロより小さい値が指定された場合は、デフォルトの 4096 が使用されます。

noRec ソケットのタイムアウト値 (ミリ秒) を指定する整数。

戻り値 なし

例外 なし

例

```
myLengthAdapter.activate("Network:localhost:8082",null,MQeTcpipLengthAdapter.MQe_Adapter_NOPERSIST, 4096,300000);
```

MQeTcpipLengthAdapter read

構文

```
public byte[] read( Object opt, int recordSize )
                    throws Exception {
```

説明 基礎となる TCP/IP ストリームからバイトを読み取ります。

パラメーター

opt

MQe.MQe_Adapter_HEADER
ヘッダー (データ長) 情報を読み取る

recordSize 1 回の読み取り操作で読み取るバッファー・サイズの最大値を指定する整数。

戻り値 指定されたバイト数を最大値としたバイト配列のデータ。ブロックは、要求されたバイト数よりも小さい配列になる場合があります。

例外

EOFException
データがこれ以上使用できない場合

例

```
byte[] results = myLengthAdapter.read(MQe.MQe_Adapter_HEADER, 4096);
```

MQeTcpipLengthAdapter write

構文

```
public void write( Object opt, int recordSize, byte data[] )
                   throws Exception
```

説明 指定されたバイト配列を TCP/IP 出力ストリームに書き込みます。なお、送信側に不必要なデータを残さないため、書き込みの終わったデータ・ストリームはフラッシュされます。

パラメーター

opt

MQe.MQe_Adapter_HEADER
ヘッダー (データ長) 情報を設定するストリング

recordSize 書き込むバイト数を指定する整数

data 書き込まれるデータ

戻り値 なし

例外 なし

例

```
myLengthAdapter.write(MQe.MQe_Adapter_HEADER, 4096, myByteArray);
```

MQeTcpipHistoryAdapter

MQeTcpipHistoryAdapter は、 MQeTcpipLengthAdapter の上に特別なプロトコルの層を加え、永続的なソケットの接続と最近使用したデータを短縮された形式でエンコードする機能をサポートします。このため、MQeTcpipHistoryAdapter は、全般的なパケット・サイズに関して非常に効果的な通信メカニズムとなっています。

ヒストリーをエンコードするオプションでチャネル・レベルの暗号化や圧縮を使用する際には注意が必要です。送信側のアダプターと受信側のアダプターとの間でヒストリー・データが異なっている場合があるからです (たとえば、暗号鍵が変わっている場合など)。

パッケージ **com.ibm.mqe.adapter**

このクラスは、MQeTcpipLengthAdapte の下位クラスです。ここにリスト表示されていない通常のアダプター操作はすべて、このスーパークラスで提供される実装を使用します。

メソッド

メソッド	目的
activate	新規の MQeTcpipHistoryAdapter オブジェクトを初期設定します。
close	MQeTcpipHistoryAdapter オブジェクトをクローズします。

MQeTcpipHistoryAdapter activate

構文

```
public void   activate( String  fileDesc,
                        Object  param,
                        Object  options,
                        int     lrecl,
                        int     noRec ) throws Exception
```

説明 新規の MQeTcpipHistoryAdapter オブジェクトを初期設定します。

パラメーター

fileDesc このアダプターを表すストリング
(たとえば ″Network:1.2.3.4:8082″)

param ACCEPT が出された場合に、作成されるアダプターに指定されるパラメーター

options 以下のオープン・オプションを指定するストリング

MQe.MQe_Adapter_LISTEN
listen 用ソケットを作成する

MQe.MQe_Adapter_ACCEPT
接続 MQe.MQe_Adapter_LOCALHOST を受け入れ、ローカル・ホスト・アドレスを戻す

MQe.MQe_Adapter_NETWORK
ネットワーク・タイプ (TCPIP) を戻す

MQe.MQe_Adapter_PERSIST
ソケットの永続性を使用可能にする (永続性はデフォルトで使用可能になっているので、このオプションは使用しないでください。)

MQe.MQe_Adapter_QOSINPUTS
有効な「サービス品質 (QOS) (Quality of Service (QOS))」名を戻す

MQe.MQe_Adapter_SETSOCKET
予約済み

MQeTcpipHistoryAdapter.MQe_Adapter_HISTORY
ヒストリー機能を使用する。(ヒストリーはデフォルトで使用可能になっているので、このオプションは使用しないでください。)

MQeTcpipHistoryAdapter.MQe_Adapter_NOHISTORY
ヒストリー機能を使用不可にする。

MQeTcpipLengthAdapter.MQe_Adapter_NOPERSIST
永続ソケットを使用不可にする。

たとえば、永続性やヒストリーのサポートを使用不可にするために、ヒストリー・アダプターを構成する場合は、ここで、 `MQeTcpipLengthAdapter.MQe_Adapter_NOPERSIST + MQeTcpipHistoryAdapter.MQe_Adapter_NOHISTORY` を指定します。これは、ストリング `"<NOPERSIST><NOHISTORY>"` と同じ意味を持ちます。

lrecl TCP/IP ブロック・サイズを指定する整数。ゼロより小さい値が指定された場合は、デフォルトの 4096 が使用されます。

noRec ソケットのタイムアウト値 (ミリ秒) を指定する整数。

戻り値 なし

例外 なし

例

```
myHistoryAdapter.activate("Network:localhost:8082", null,
                          MQe.MQe_Adapter_PERSIST +
                          MQeTcpipHistoryAdapter.MQe_Adapter_HISTORY,
                          4096, 300000);
```

MQeTcpipHistoryAdapter close

構文

```
public void close( Object opt ) throws Exception {
```

説明 現在のアダプターをクローズします。これにより、このアダプターに対して読み取りや書き込みを行うことはできなくなります。

パラメーター

opt

MQe.MQe_Adapter_HEADER
これがこのアダプターの最後のクローズの場合は、MQe.MQe_Adapter_FINAL を含むストリング。この場合、ヒストリーのバッファーはすべて空になります。

戻り値 なし

例外 なし

例

```
myHistoryAdapter.close(MQe.MQe_Adapter_FINAL);
```

MQeUdpipAdapter

MQeUdpipAdapter は、UDP/IP データグラムにおける確実なデータ転送をサポートします。

パッケージ **com.ibm.mqe.adapter**

このクラスは、MQeAdapter の下位クラスです。

メソッドの要約

メソッド	目的
activate	新規の MQeUdpipAdapter オブジェクトを初期設定します。
close	現在の MQeUdpipAdapter オブジェクトをクローズします。
control	現在のアダプター・インスタンスに関する情報を設定または取得します。
open	使用するアダプターをオープンします。
read	アダプターからバイトを読み取ります。
status	アダプターに関する情報を戻します。
write	アダプターにバイトを書き込みます。

MQeUdpipAdapter activate

構文

```
public void activate(String fileDesc,
                     Object param,
                     Object options,
                     int blocksize,
                     int timeout) throws Exception
```

説明 活動化メソッドは、アダプターを初期化します。

MQeAdapter クラスの **activate()** をオーバーライドします。

パラメーター

fileDesc このアダプターを表すストリング。

param ACCEPT が出された場合に、作成されるアダプターに指定されるパラメーター・オブジェクト

options 使用する初期オプション。次のいずれかまたは両方です。

MQe.MQe_Adapter_LISTEN
listen 用ソケットを作成する

MQe.MQe_Adapter_ACCEPT
接続を受け入れる

blocksize 使用する UDP/IP ブロック・サイズを指定する整数 (デフォルトは 4096)

timeout タイムアウト値 (ミリ秒) を指定する整数。

戻り値 なし

MQeUdpipAdapter close

構文

```
public void close( Object opt ) throws Exception
```

説明 このメソッドは、アダプターをクローズします。 MQe_Adapter_PERSIST オプションを設定してアダプターをオープンした場合は、 **close()** メソッドの呼び出しに、 MQe_Adapter_FINAL オプションを含めない限り、クローズは有効にならないので注意してください。

MQeAdapter クラスの **close()** をオーバーライドします。

パラメーター

opt クローズで使用するオプション - MQeAdapter として

戻り値 なし

MQeUdpipAdapter control

構文

```
public Object control(Object opt, Object ctrlobj ) throws Exception
```

説明 このメソッドは、現在のアダプター・インスタンスに関する情報を設定または取得します。

MQeAdapter クラスの **control()** をオーバーライドします。

パラメーター

opt 照会または設定のオプションで、以下のいずれかを含むもの

MQe.MQe_Adapter_ACCEPT
接続を待機し、受け入れる (listen 用ソケット)

MQe.MQe_Adapter_SETSOCKET
提供されたソケット・オブジェクトを使用する

MQe.MQe_Adapter_PULSE
タイマー・パルス・スレッドを開始する

MQe.MQe_Adapter_QOSINPUTS -
「サービス品質 (QOS) (Quality of Service (QOS))」入力パラメーターのリストを戻す

ctrlObj 一部のオプションが使用するパラメーター。オブジェクト

は、ネットワーク定義を識別するストリング (MQe_Adapter_ACCEPT) か、アダプターがパケットの送信に使用するソケット (MQe_Adapter_SETSOCKET) のいずれかです。

戻り値 MQe_Adapter_QOSINPUTS の場合は、サービスの品質パラメーターの列挙型を含むオブジェクト。

MQeUdpipAdapter open

構文

```
public void open( Object opt ) throws Exception
```

説明 このメソッドは、ソケット (必要な場合) を作成し、使用するアダプターを準備します。

MQeAdapter クラスの **open()** をオーバーライドします。

パラメーター

opt 使用しない

戻り値 なし

MQeUDPAdapter read

構文

```
public byte[] read( Object opt, int recordSize )
                    throws Exception
```

説明 このメソッドは、アダプターのターゲット・システムからデータを読み取ります。

MQeAdapter クラスの **read()** をオーバーライドします。

パラメーター

opt 読み取りで使用するオプション。

recordSize 読み取るバイト数を指定する整数

戻り値 読み取られたデータ。

MQeUdcpipAdapter status

構文

```
public String status( Object  opt  ) throws Exception
```

説明 このメソッドは、現在のアダプター・インスタンスに関する情報を取得します。

MQeAdapter クラスの **status()** をオーバーライドします。

パラメーター

opt 以下のどちらが必要かを照会するオプションを含むストリング。

MQe.MQe_Adapter_NETWORK
使用中のネットワークのタイプ (UDPIP) を取得する

MQe.MQe_Adapter_LOCALHOST
ローカル・ホスト名を取得する

MQeAdapter で有効なオプションも継承されます。

戻り値 ストリングとして要求される情報

MQeUdpipAdapter write

構文

```
public void write( Object opt, int length, byte data[] )
                   throws Exception
```

説明 このメソッドは、指定のバイト配列をターゲット・システムに送信します。

MQeAdapter クラスの **write()** をオーバーライドします。

パラメーター

opt 書き込みに使用するオプションを含むストリング。

length 書き込むバイト数を指定する整数

data 送信するデータ

戻り値 なし

MQeWESAuthenticationAdapter

このアダプターは、Websphere Everyplace 認証プロキシーおよび透過プロキシーを介した、 HTTP 要求のトンネルをサポートします。これは、com.ibm.mqe.adapters.MQeTcpipHttpAdapter を拡張する通信アダプターで、それ自体はこの (あるいは他の任意の) 通信アダプターの代わりとして使用できます。このアダプターは、静的メソッド **setBasicAuthorization()** を使用する MQSeries Everyplace アプリケーションを提供します。ユーザーは、このメソッドにユーザー ID とパスワードのストリングを組み込むことができます。一度設定すると、以後**書き込み**呼び出しを行うたびに、クラスが HTTP ヘッダーに許可の行を追加します。この許可の行には、アダプターによって Base64 にエンコードされるユーザー ID およびパスワード情報が入っています。たとえば、以下のようになっています。

```
Authorization: Basic QmFzZTY0OmlzTm90RW5jcnlwdGVk
```

アダプター **write** に提供されている HTTP ヘッダーに、すでに許可の行が含まれている場合は、その既存の行が使用されます。基本的な Web 権限では、ユーザーのクリデンシャルが HTTP ヘッダー内部で Base64 にエンコードされている必要があるので、注意してください。これは暗号化されたフォーマットではありません。 Base64 はクリア・テキストのエンコード方式で、その内容は、 HTTP フローにアクセスできる人ならだれでも見ることができます。 Base64 情報を暗号化しないままでネットワークに送信したくない場合は、 MQSeries Everyplace メッセージ・レベル・セキュリティーなどのセキュリティー・ソリューションを提供する必要があります。

MQeWESAuthenticationAdapter は、アダプター **reads** の代行受信も行い、サーバーからの次のような無効応答をチェックします。

- 「無許可 (Unauthorized)」(401)、「禁止 (Forbidden)」(403)、または「プロキシー認証が必要 (Proxy Authentication Required)」(407) 応答。アダプターは、タイプ `MQe.Except_Authenticate` の MQeException を、アプリケーションに再び throw します。
- 「見つかりません (Not found)」応答 (404)。タイプ `MQe.Except_NotFound` の MQeException が throw されます。
- サーバー・エラー (50x) およびその他のクライアント・エラーすべて (40x)。タイプ `MQe.Except_NotSupported` の MQeException が throw されます。

すべての例外のテキストはヘッダー行で、そこには、実際の応答タイプ (たとえば ″HTTP/1.1 500 Internal Server Error″) が含まれています。

失敗の応答があると、アダプターは、戻された応答ヘッダーの領域情報をスキャンします。この情報は、 MQeWESAuthenticationAdapter で静的な **getRealm()** メソッドを使用すれば、後で入手することもできます。

パッケージ　　com.ibm.mqe.adapter

このクラスは、 MQeTcpipHttpAdapter の拡張クラスです。

メソッド

メソッド	目的
activate	アダプターを初期化します。
getDeviceName	現在の発信装置名を戻します。
getRealm	サーバーの呼びかけに基づいて、領域情報を戻します。
read	WES からのデータを読み取ります。
readEOF	HTTP ストリームからできるだけ多くのものを読み取ります。
readln	データ行を読み取ります。
setBasicAuthorization	HTTP フローに送信する前に Base64 にエンコードする必要のある、ユーザー ID とパスワードを設定します。
setDeviceName	発信装置の名前を、 HTTP 要求の「ユーザー - エージェント (User-Agent)」フィールドで使用するとおりに設定します。
write	HTTP ストリームにバイトを書き込みます。
writeln	テキスト行をアダプターに送ります。

MQeWESAuthenticationAdapter activate

構文

```
public void activate(String fileDesc,
                     Object param,
                     Object options,
                     int lrecl,
                     int noRec) throws Exception
```

説明 このメソッドは、アダプターを初期化します。アダプターが透過プロキシーとして使用されている場合は、そのアダプターのプロキシー設定もセットアップされます。そうでない場合は、このメソッドは、 MQeTcpipAdapter と同じ働きをします。

MQeTcpipAdapter クラスに **activate** をオーバーライドします。

パラメーター

fileDesc HTTP ターゲットの URL を表すストリング。たとえば `http://mqe.hursley.ibm.com:8082`。

param ACCEPT が出された場合に、作成されるアダプターに指定されるパラメーター

options 以下のオープン・オプションを指定するオブジェクト

MQe.MQe_Adapter_LISTEN
listen 用ソケットを作成する

MQe.MQe_Adapter_ACCEPT
接続を受け入れる

MQe.MQe_Adapter_LOCALHOST
ローカル・ホスト・アドレスを戻す

MQe.MQe_Adapter_NETWORK
ネットワーク・タイプ (TCPIP) を戻す

MQe.MQe_Adapter_PERSIST
呼び出しの受け入れ時に永続オブジェクトをコピーする

MQe.MQe_Adapter_QOSINPUTS
有効な「サービス品質 (QOS) (Quality of Service (QOS))」名を戻す

MQe.MQe_Adapter_SETSOCKET
予約済み

lrecl TCP/IP ブロック・サイズを指定する整数。ゼロより小さい値が指定された場合は、デフォルトの 4096 が使用されます。

norec ソケットのタイムアウト値 (ミリ秒) を指定する整数。

戻り値 なし

例外 なし

MQeWESAuthenticationAdapter setBasicAuthorization

構文

```
public static void setBasicAuthorization(String userid,
                                         String password)
```

説明 setBasicAuthorization メソッドは、 Base64 にエンコードしてから HTTP フローに送信するユーザー ID とパスワードを設定します。

パラメーター

userid ユーザー ID を含むストリングです。

password ユーザー・パスワードを含むストリングです。

戻り値 なし

例外 なし

MQeWESAuthenticationAdapter setDeviceName

構文

```
public static void setDeviceName(String name)
```

説明 発信装置の名前を、 HTTP 要求数の「ユーザー - エージェント (User-Agent)」フィールドで使用するとおりに設定します。

パラメーター

name 装置名を表すストリング。

戻り値 なし

例外 なし

MQeWESAuthenticationAdapter getDeviceName

構文

```
public static String getDeviceName()
```

説明 現在の発信装置名を戻します。

パラメーター

なし

戻り値 装置名

例外 なし

MQeWESAuthenticationAdapter read

構文

```
public byte[] read(Object opt,
                   int recordSize) throws Exception
```

説明 Websphere Everyplace からのデータを読み取ります。認証サーバーは、常に標準の HTTP 応答を送信するので、スーパークラスを使用すれば読み取ることができます。ただし、このメソッドは失敗の応答コードを解釈して、それを適切な MQeException に変換します。

MQeTcpipAdapter クラスの **read()** をオーバーライドします。

パラメーター

opt 読み取り対象のオブジェクトを指定するストリング。

MQe_Adapter_HEADER
HTTP ヘッダーを読み取る

MQe_Adapter_CONTENT
内容データを読み取る

recordSize 読み取るバイト数を指定する整数

戻り値 読み取られたデータを含むバイト配列。

例外

MQeException

- 「無許可 (Unauthorized)」(401)、「禁止 (Forbidden)」(403)、または「プロキシー認証が必要 (Proxy Authentication Required)」(407) 応答の場合は、 `MQe.Except_Authenticate`

- 「見つかりません (not found)」応答 (404) の場合は MQe.Except_NotFound
- サーバー・エラー (50x) およびその他のクライアント・エラーすべて (40x) の場合は、 MQe.Except_NotSupported

MQeWESAuthenticationAdapter readEOF

構文

```
public byte[] readEOF(Object opt) throws Exception
```

説明 HTTP ストリームからできるだけ多くを読み取ります。

MQeAdapter クラスの **readEOF()** をオーバーライドします。

パラメーター

opt 読み取り対象のオブジェクトを指定するストリング。

MQe_Adapter_HEADER
HTTP ヘッダーを読み取る

MQe_Adapter_CONTENT
内容データを読み取る

戻り値 読み取られたデータ

MQeWESAuthenticationAdapter readln

構文

```
public String readln(Object opt) throws Exception
```

説明 Websphere Everyplace からのデータ行を読み取ります。

MQeTcpipAdapter クラスの **readln()** をオーバーライドします。

パラメーター

opt 読み取り対象のオブジェクトを指定するストリング。

MQe_Adapter_HEADER
HTTP ヘッダーを読み取る

MQe_Adapter_CONTENT
内容データを読み取る

戻り値 読み取られたデータ

MQeWESAuthenticationAdapter write

構文

```
public synchronized void write(Object opt,
                               int recordSize,
                               byte data[]) throws Exception
```

説明 HTTP ストリームにバイトを書き込みます。WWW 認証行 (透過プロキシーの場合は、プロキシー認証行) をユーザー・データに追加してから、それを基礎となるアダプターに渡します。

MQeTcpipAdapter クラスの **write()** をオーバーライドします。

パラメーター

opt 次のいずれかの値を含むストリング

MQe_Adapter_HEADER
'add an HTTP POST' 要求をデータの先頭に書き込む

MQe_Adapter_HEADERSP
HTTP 応答ヘッダーを書き込む

オプションが指定されない場合、データはそのまま送信されます。

recordSize 書き込まれるデータの長さを指定する整数。すべてを書き込む場合は -1。

data 書き込まれるデータ

戻り値 フィルター名またはファイル名のいずれかを含むストリング

例外 なし

例

```
String filename = (String) mydfa.status(MQe.MQe_Adapter_FILENAME);
```

MQeWESAuthenticationAdapter writeln

構文

```
public void writeln(Object opt,
                    String data) throws Exception
```

説明 テキスト行をアダプターに送ります。

MQeTcpipAdapter クラスの **writeln()** をオーバーライドします。

パラメーター

opt 次のいずれかの値を含むストリング

MQe_Adapter_HEADER
HTTP GET 要求の行をラップします。

MQe_Adapter_HEADERSP
HTTP 応答の行をラップします。

data 書き込む行。

戻り値 なし

MQeWESAuthenticationAdapter getRealm

構文

```
public static String getRealm()
```

説明 サーバーの呼びかけに基づいて、領域情報を決定します。

パラメーター
なし

戻り値 認証が必要な領域を表すストリング、または、領域が決定できない場合はブランク・ストリング。

例外 なし

第2部 例外および戻りコード

第10章 例外

この章では、MQSeries Everyplace から throw される可能性のある例外を詳しく紹介します。前半は、数値順の例外のリストです。そして後半は、詳しい説明付きで例外をアルファベット順にリストしています。

数値順

- 002 - Except_NotSupported
- 003 - Except_Syntax
- 004 - Except_Type
- 006 - Except_NotFound
- 007 - Except_Data
- 008 - Except_BadRequest
- 009 - Except_Stopped
- 010 - Except_Closed
- 011 - Except_Duplicate
- 012 - Except_NotAllowed
- 013 - Except_Rule
- 014 - Except_Timeout
- 020 - Except_Chnl_Attributes
- 022 - Except_Chnl_Destination
- 023 - Except_Chnl_Limit
- 024 - Except_Chnl_ID
- 025 - Except_Chnl_Overrun
- 041 - Except_Trnsport_Request
- 100 - Except_QMgr_NotActive
- 101 - Except_QMgr_InvalidQMgrName
- 102 - Except_QMgr_Activated
- 103 - Except_QMgr_AlreadyExists
- 104 - Except_QMgr_InvalidQName
- 105 - Except_QMgr_QExists
- 106 - Except_QMgr_UknownQMgr
- 107 - Except_QMgr_QNotEmpty
- 108 - Except_QMgr_QDoesNotExist

- 110 - Except_QMgr_WrongQtype
- 113 - Except_QMgr_NotConfigured
- 121 - Except_Q_NoMatchingMsg
- 124 - Except_Q_InvalidPriority
- 125 - Except_Q_Full
- 126 - Except_Q_MsgTooLarge
- 127 - Except_Q_NotActive
- 128 - Except_Q_Active
- 129 - Except_Q_InvalidName
- 130 - Except_Q_TargetRegistryRequired
- 231 - Except_Con_AliasAlreadyExists
- 232 - Except_Con_AdapterRequired
- 233 - Except_Con_InvalidName
- 301 - Except_Reg_NullName
- 302 - Except_Reg_AlreadyExists
- 303 - Except_Reg_DoesNotExist
- 304 - Except_Reg_OpenFailed
- 305 - Except_Reg_InvalidSession
- 306 - Except_Reg_NotDefined
- 309 - Except_Reg_AddFailed
- 310 - Except_Reg_DeleteFailed
- 311 - Except_Reg_ReadFailed
- 312 - Except_Reg_UpdateFailed
- 313 - Except_Reg_ListFailed
- 314 - Except_Reg_SearchFailed
- 315 - Except_Reg_RenameFailed
- 316 - Except_Reg_ResetPINFailed
- 317 - Except_Reg_CRTKeyDecFailed
- 318 - Except_Reg_CRTKeySignFailed
- 319 - Except_Reg_DeleteRegistryFailed
- 320 - Except_Reg_AlreadyOpen
- 321 - Except_Reg_NotSecure
- 350 - Except_PrivateReg_BadPIN
- 351 - Except_PrivateReg_ActivateFailed
- 361 - Except_MiniCertreg_ActivateFailed

- 370 - Except_PublicReg_ActivateFailed
- 371 - Except_PublicReg_InvalidRequest
- 400 - Except_Admin_NotAdminMsg
- 401 - Except_Admin_ActionNotSupported
- 402 - Except_Admin_InvalidField
- 500 - Except_Authenticate
- 501 - Except_S_Cipher
- 502 - Except_S_InvalidSignature
- 503 - Except_S_CertificateExpired
- 504 - Except_S_InvalidAttribute
- 505 - Except_S_MiniCertNotAvailable
- 506 - Except_S_RegistryNotAvailable
- 507 - Except_S_BadIntegrity
- 508 - Except_S_NoPresetKeyAvailable
- 509 - Except_S_MissingSection
- 510 - Except_S_BadSubject
- 600 - Except_Generic_MQ_Error
- 601 - Except_MQ_QMgr_Not_Available
- 602 - Except_Characteristic_Unknown
- 603 - Except_Characteristic_Read_Only_Violation
- 604 - Except_Bridge_Not_On_PreLoad_List
- 605 - Except_Registry_Update_Failed
- 606 - Except_Characteristic_Not_Found_In_Input
- 608 - Except_Invalid_Registry_Entry_Name
- 609 - Except_Not_Transmission_Queue
- 610 - Except_Bridge_Internal_Error
- 611 - Except_MQ_QMgr_InvalidName
- 612 - Except_MQ_Q_InvalidName
- 613 - Except_Lost_MQ_Q_Handle
- 614 - Except_Resource_Not_Found
- 615 - Except_Transformation_Error
- 616 - Except_Parent_Not_Running
- 617 - Except_Invalid_Run_State
- 618 - Except_Could_Not_Delete_Myself
- 619 - Except_Listener_State_Error

- 620 - Except_Invalid_Characteristic
- 621 - Except_Delete_Children_First
- 622 - Except_InvalidName
- 623 - Except_Class_Not_MQeRules
- 625 - Except_Object_Not_Started
- 626 - Except_ShutDown_Children_First
- 627 - Except_Invalid_Reg_Type
- 629 - Except_Message_Undelivered
- 630 - Except_Bridges_Already_Loaded
- 631 - Except_No_Transformer_Found
- 632 - Except_Lost_MQ_Connection
- 633 - Except_Client_Con_Not_Available
- 634 - Except_Exit_Class_Init_Error

アルファベット順

Except_Admin_ActionNotSupported - 401

説明: このアクションは、基本クラスではサポートされていません。ブロック化したこれらのメソッドは、サブクラスでオーバーライドされます。

ユーザー処置: この状態は、 MQSeries Everyplace が内部で処理するはずなので、問題にはなりません。

Except_Admin_InvalidField - 402

説明: サポートされていないフィールドが、 MQeAdminMsg 要求で指定されました。

ユーザー処置: サポート対象の MQeAdminMsg フィールドは、特性メソッドによって使用可能です。

Except_Admin_NotAdminMsg - 400

説明: 管理キューに届いたメッセージが、 MQeAdminMsg の下位クラスではありません。

ユーザー処置: この状態は、 MQSeries Everyplace が内部で処理するはずなので、問題にはなりません。

Except_Authenticate - 500

説明: アクションの認証に提供された情報に、障害が発生しました。

ユーザー処置: 提供されたクリデンシャルが有効であることを確認して、要求を繰り返します。

Except_BadRequest - 008

説明: 暗号化 / 暗号化解除鍵が設定されていません。あるいは、正しくない 16 進データが変換のために渡されました。

ユーザー処置: 関連したテキストで、ガイダンスが提供されます。

Except_Bridge_Internal_Error - 610

説明: 予期しないエラーが発生しました。

ユーザー処置: お手近のサポート担当者に連絡してください。

Except_Bridge_Not_On_PreLoad_List - 604

説明: ブリッジ・リソースにアクセスしようとしましたが、 MQbridges オブジェクトがロードされていません。

ユーザー処置: MQBridge クラス別名をセットアップして、サーバーを再始動してからでないと、MQSeries Everyplace サーバー上のブリッジ・リソースは使用できません。また、サーバー・プログラムで、ブリッジをロードして活動化する必要もあります。

Except_Bridges_Already_Loaded - 630

説明: 2 つの MQeBridges オブジェクトを、 1 つの JVM にインスタンス化しようとしました。これは許可されていません。

ユーザー処置: 構成がこれを許可していないことを確認します。

Except_Characteristic_Not_Found_In_Input - 606

説明: MQSeries-Bridge のリソースが、 MQeFields オブジェクトで特定の特性を検出しようとしましたが、 MQeFields オブジェクトにはありませんでした。この MQSeries-Bridge リソースのレジストリー項目の一部が除去されている場合に、こうなることがあります。

ユーザー処置: MQeFields オブジェクトから、有効な特性にアクセスしようと試みるだけです。

Except_Characteristic_Read_Only_Violation - 603

説明: 設定できない特性を設定しようとしました。たとえば、オブジェクトの名前は変更できません。その実行状態も、スタート・ストップ管理メッセージを使用しなければ変更できません。

ユーザー処置: 特性を設定するアプリケーションを修正して、読み取り専用属性を設定しないようにします。

Except_Characteristic_Unknown - 602

説明: Inquire または InquireAll 管理メッセージをブリッジに送信して、特性を照会しようとしました。照会した特性は、Inquire メッセージの宛先のオブジェクトには知られていません。

ユーザー処置: 管理メッセージを生成するアプリケーションを、有効な特性だけを照会するように修正します。

Except_Chnl_Attributes - 020

説明: MQSeries Everyplace チャネル属性のミスマッチ。

ユーザー処置: この例外と共に表示されるテキストで、この問題が発生した理由が十分わかるはずです。

Except_Chnl_Destination - 022

説明: MQSeries Everyplace チャネルの宛先が認識されません。

ユーザー処置: MQSeries Everyplace 接続定義が正しいかどうかを確認します。指定されたアダプターが MQeAdapter の下位クラスはないか、あるいは、ファイル記述子が間違っていることが考えられます。

Except_Chnl_ID - 024

説明: MQSeries Everyplace チャネルは使用できなくなりました。

ユーザー処置: リモート・キュー・マネージャーが、コンタクト可能であることを確認します。リモート・キュー・マネージャーがこのチャネルをクローズしたのかどうかを突き止めます。

Except_Chnl_Limit - 023

説明: チャネルの最大数を超過しました。

ユーザー処置: キュー・マネージャー・スタートアップ ini ファイルの MaxChannels 設定を調整して、チャネルの最大数を増やすことを検討します。

Except_Chnl_Overrun - 025

説明: MQePeerChannel が 2 つの応答を受信しました。

ユーザー処置:

Except_Class_Not_MQeRules - 623

説明: MQeRules からの継承ではないルール・クラスが呼び出されました。

ユーザー処置: すべてのルールは、 com.ibm.MQe.MQeRules からの継承でなければなりません。つまり、**Permit()** メソッドをインプリメントしなければなりません。構成情報で指定したルール・クラスが有効であることを確認してください。 MQSeries-Bridge で使用されるルール・クラスの完全なリストとその使用箇所については、文書内の別の場所を参照してください。

Except_Client_Con_Not_Available - 633

説明: 存在しないクライアント接続オブジェクトに、アクセスしようとしました。

ユーザー処置: MQSeries-Bridge オブジェクトが、有効なクライアント接続オブジェクトを指していることを確認します。必要なら、クライアント接続が存在していてアクセス可能であるかどうかを照会します。

Except_Closed - 010

説明: 通信チャネルがクローズしています。

ユーザー処置: この状態は、 MQSeries Everyplace が内部で処理するはずなので、問題にはなりません。 MQSeries Everyplace のピアツーピア・ソリューションを開発している場合は、この例外を処理しなければならないことがあります。

Except_Con_AdapterRequired - 232

説明: チャネルを指定する場合はアダプターが必要です。

ユーザー処置: チャネルとともに使用するアダプターを指定します。

Except_Con_AliasAlreadyExists - 231

説明: 接続の別名がすでに存在しています。

ユーザー処置: この接続別名はすでに定義されているので、新規の別名を選択します。この例外を処理しなければならないことがあります。

Except_Con_InvalidName - 233

説明: 接続名が命名基準に準拠していません。

ユーザー処置: 接続名は 1 文字以上の ASCII ストリングで、値は 20 から 128 までです。使用できる文字は以下のとおりです。

- 大文字 A 〜 Z
- 小文字 a 〜 z
- 数値 0 〜 9
- ピリオド (.)
- 下線 (_)
- パーセント記号 (%)

最初と最後の文字にはピリオド (.) は使えません。

Except_Could_Not_Delete_Myself - 618

説明: 予期しないエラーが発生しました。

ユーザー処置: お手近のサポート担当者に連絡してください。

Except_Data - 007

説明: MQSeries Everyplace がデータに問題を検出すると、この例外が発生します。暗号機能または何らかの破壊に関する問題などが考えられます。

ユーザー処置: この例外と共に表示されるテキストで、この問題が検出された理由が十分わかるはずです。

Except_Delete_Children_First - 621

説明: ユーザーが、 MQSeries-Bridge の親子階層の中の管理対象オブジェクトを、その子オブジェクトを先に削除しないで削除しようとしました。これはまた、″effect-children″ フラグが設定されていないために、削除イベント

がまず管理対象オブジェクトの子オブジェクトに伝わらないということをも意味します。管理対象オブジェクトを削除するには、まず、その子オブジェクトを削除しなければなりません。

ユーザー処置: 管理対象オブジェクトの子オブジェクトを、すべて個別に削除するか、″effects-children″ を使用して、この管理対象オブジェクトを削除するのと同じアクションで、その子オブジェクトをすべて削除するかします。

Except_Duplicate - 011

説明: MQeFields オブジェクトがコピーされ、重複項目が検出されました。

ユーザー処置: MQeFields.copy メソッドで、置換パラメーターを `true` に設定します。

Except_Exit_Class_Init_Error - 634

説明: クライアント接続オブジェクトの一部として定義されたユーザー出口を、初期化できませんでした。

ユーザー処置: ご使用の出口が、 Java ユーザー出口インターフェースの MQSeries クラスに準拠していることを確認します。詳しくは、「MQSeries Using Java」を参照してください。

Except_Generic_MQ_Error - 600

説明: MQSeries Everyplace サーバーと通信する際に、 Java の MQSeries クラスがエラー・コードを戻しました。MQSeries 理由コードは、例外に組み込まれています。

ユーザー処置: このエラーの解釈方法の詳細については、「*MQSeries アプリケーション・プログラミング・リファレンス*」を参照してください。

Except_Invalid_Characteristic - 620

説明: ユーザーが、更新管理メッセージまたは作成管理メッセージを介して、特性の値を設定しようとしている可能性があります。たとえば、値が大きすぎるか小さすぎます。

ユーザー処置: 特定の特性の値の範囲を調べ、要求を修正してからもう一度試みます。

Except_InvalidName - 622

説明: MQSeries-Bridge の名前に無効文字が含まれていました。あるいは名前が長すぎるか短すぎます。

ユーザー処置: 40 文字より短く、英数字だけでできているブリッジ名を選択します。

Except_Invalid_Registry_Entry_Name - 608

説明: MQSeries-Bridge が、そのオブジェクトの 1 つのレジストリー項目名 (おそらくユーザーからの入力に基づいたもの) を、管理メッセージを介して作成しようとしました。この名前は、レジストリーが設定したルールに準拠していません。

ユーザー処置: レジストリー項目名に無効な文字がないかどうか、また名前が長すぎないかどうかを調べます。修正して、再度試みます。

Except_Invalid_Reg_Type - 627

説明: レジストリーが、使用されたレジストリー・タイプを認識しませんでした。

ユーザー処置: MQSeries Everyplace レジストリーは、″レジストリー・タイプ″の概念を使用して、それが現在処理しているレジストリー・レコードの種類を示します。同じタイプの項目は、同じファイル・ディレクトリーに保管されています。このレジストリー・タイプが有効であることを確認してください。

Except_Invalid_Run_State - 617

説明: 予期しないエラーが発生しました。

ユーザー処置: お手近のサポート担当者に連絡してください。

Except_Listener_State_Error - 619

説明: リスナーが、状態のロギング情報の読み取りまたは書き込みを行おうとしているときに、エラーが発生しました。

ユーザー処置: リスナー状態保管クラスが正しいこと、状態ファイルが読み取りおよび書き込み可能 (MQeDiskFieldsAdapter の場合) か、 sync キューがアクセス可能で put および get が使用可能になっている (MQeMQAdapter の場合) ことを、検証します。

Except_Lost_MQ_Connection - 632

説明: MQeMQAdapter が、 MQSeries に対して有効な処理を行うことができませんでした。

ユーザー処置: これまでの任意の情報を検討して、 MQSeries への接続が失敗した理由を突き止めます。問題を訂正して再試行します。

Except_Lost_MQ_Q_Handle - 613

説明: MQSeries オープン・キューの参照が、突然無効になりました。

ユーザー処置: MQSeries への接続が失敗した状況と理由を突き止めます。MQBridge オブジェクトの再始動を試みます。

Except_Message_Undelivered - 629

説明: リスナーを、メッセージが配達されないため (ルールにより) 停止するように指定されました。または、リスナーは、書き込み中に指定されないエラーが起こってから、この MQeException を、未配布メッセージ・ルール・クラスに渡すこともできます。

ユーザー処置: この例外が throw されないようにするには、未配布メッセージ・ルールを修正します。

Except_MiniCertreg_ActivateFailed - 361

説明: いくつかの理由で、活動化に失敗することがあります。

- keyRingPassword が提供されなかった場合
- 認証要求が失敗した場合
- 登録プロセスが失敗した場合

ユーザー処置: この例外と共に表示されるテキストに、詳しい説明があります。

Except_MQ_Q_InvalidName - 612

説明:

ユーザー処置:

Except_MQ_QMgr_InvalidName - 611

説明: MQSeries キュー・フィールドに無効文字が含まれていました。あるいはキュー・フィールドが長すぎるか短すぎます。

ユーザー処置: アクセスしようとしている MQSeries キュー・マネージャーの名前が、 MQSeries Everyplace で構成されている値に一致していることを確認します。

Except_MQ_QMgr_Not_Available - 601

説明: MQSeries サーバーと通信する際に、 Java クライアントがエラー・コードを戻しました。戻されたエラーは、 MQSeries キュー・マネージャーが使用できないことを示唆しているようです。 MQSeries 理由コードは、例外に組み込まれています。

ユーザー処置: このエラーの解釈方法の詳細については、「*MQSeries アプリケーション・プログラミング・リファレンス*」を参照してください。

Except_No_Transformer_Found - 631

説明: デフォルトのトランスフォーマーの名前が見つかりませんでした。

ユーザー処置: MQeMQBridgeQueue 定義内で使用したい、トランスフォーマーのクラス名を設定します。また、デフォルトのトランスフォーマーを、ブリッジの構成内に設定してください。

Except_NotAllowed - 012

説明: MQSeries Everyplace が要求やリソースの状態に関して問題を検出すると、この例外が発生します。

ユーザー処置: この例外と共に表示されるテキストで、この問題が検出された理由が十分わかるはずです。

Except_NotFound - 006

説明: オブジェクトが見つかりません。たとえば、それがメッセージやキュー・マネージャー、フィールド、あるいは接続定義の場合もあります。この例外が発生する可能性があるのは、MQeWESAuthenticationAdapter を使用したときに、要求されたリソースをプロキシーが見つけられなかった場合です。

ユーザー処置: この例外に伴う追加テキストによって、この状態が理解できるはずです。問題を修正してもう一度試みてください。

Except_NotSupported - 002

説明: この例外が throw されるのは、無効なメソッドまたは操作がオブジェクトのために呼び出される場合です。この例外は、MQeWESAuthenticationAdapter を使用したときに、プロキシーが認識されないエラーを throw する場合にも発生する可能性があります。

ユーザー処置: クラス・オブジェクトに不適切なメソッドを呼び出す理由を突き止めます。たとえば、putMessage は MQeHomeServerQueue、MQeRemoteQueue、および MQeStoreAndForwardQueue オブジェクトではサポートされないので、その結果、この例外が throw されることになります。問題を修正してもう一度試みてください。

Except_Not_Transmission_Queue - 609

説明: MQBridge リスナーに、送信キューではない MQSeries キューでメッセージを listen するようにとの指示が出ました。

ユーザー処置: リスナーの構成を変更して、送信キューからメッセージを取得するようにします。あるいはサーバー上のキュー定義を変更して、それを送信キューとして使用するようにします。

注: リスナー用のメッセージを書き込むアプリケーションが、この送信キューに直接書き込むことがないようにしてください。書き込みは、ターゲット・キュー・マネージャーを MQSeries Everyplace システムにしているリモートキュー定義に対して行うようにしてください。

Except_Object_Not_Started - 625

説明: 実行状態が '停止' であるときに、接続を割り振るように依頼された場合に、クライアント接続オブジェクトがこの例外を throw します。

ユーザー処置: クライアント接続オブジェクトが、 MQSeries 結合操作を実行しようとする前に開始されていることを確認します。

Except_Parent_Not_Running - 616

説明: 予期しないエラーが発生しました。

ユーザー処置: お手近のサポート担当者に連絡してください。

Except_PrivateReg_ActivateFailed - 351

説明: いくつかの理由で、活動化が失敗することがあります。

– keyRingPassword が提供されなかった場合

– 認証要求が失敗した場合

– 登録プロセスが失敗した場合

ユーザー処置: この例外と共に表示されるテキストに、詳しい説明があります。

Except_PrivateReg_BadPIN - 350

説明: 専用レジストリーには PIN が必須です。

ユーザー処置: 有効な PIN を提供します。

Except_PublicReg_ActivateFailed - 370

説明: いくつかの理由で、活動化が失敗することがあります。この例外と共に表示されるテキストに、失敗の理由が詳しく説明されています。

ユーザー処置: 問題を解決してもう一度試みます。

Except_PublicReg_InvalidRequest - 371

説明: 要求タイプが認識されませんでした。

ユーザー処置: この例外と共に表示されるテキストに、詳しい説明があります。

Except_Q_Active - 128

説明: キューは、アクティブの間は、削除することも復元することもできません。

ユーザー処置: キュー・マネージャーを停止すると、キューは非活動化されます。

Except_Q_Full - 125

説明: キューが満杯です。これは、 *MQeQueueAdminMsg.Queue_MaxQSize* で設定した値に従っています。

ユーザー処置: ソリューションに対して、キュー・サイズが十分であるかどうかを確認します。この例外を処理しなければならないこともあります。

Except_Q_InvalidName - 129

説明: キュー名が命名基準に準拠していません。

ユーザー処置: キュー名は 1 文字以上の ASCII ストリングで、値の範囲は 21 から 127 までです。使用できる文字は以下のとおりです。

– 大文字 A ～ Z

– 小文字 a ～ z、数値 0 ～ 9

– ピリオド (.)
– 下線 (_)
– パーセント記号 (%)

最初と最後の文字にはピリオド (.) は使えません。

Except_Q_InvalidPriority - 124

説明: キューのデフォルトの優先順位が有効範囲外です。

ユーザー処置: キューの優先順位を、受け入れ範囲 (0 〜 9) 以外の値に設定しようとしているかどうかを確認します。

Except_Q_MsgTooLarge - 126

説明: キューに書き込まれようとしているメッセージが、*MQeQueueAdminMsg.Queue_MaxMsgSize* に設定した値を超えています。

ユーザー処置: ソリューションに対して、このキューの最大メッセージ・サイズが十分であるかどうかを確認します。この例外を処理しなければならない場合もあります。

Except_Q_NoMatchingMsg - 121

説明: **getMessage** に、フィルターに一致するメッセージがありませんでした。

ユーザー処置: フィルターが正しく指定されていることを確認します。キューのメッセージ・ストアのメッセージを調べます。

Except_Q_NotActive - 127

説明: 初期化が正しく行われていないキューを、使用しようとしました。

ユーザー処置: キュー・パラメーターが正しいことを確認して、要求を繰り返します。

Except_Q_TargetRegistryRequired - 130

説明: オーセンティケーター・オブジェクトが、このキューに必要なレジストリーが 1 つも設定されていないことを示しています。

ユーザー処置: キュー・レジストリーの特性が、オーセンティケーター・オブジェクトで指定した特性と一致していることを確認します。

Except_QMgr_Activated - 102

説明: キュー・マネージャーが、まだアクティブであるのに復元または再活動化されようとしています。これは許可されていません。

ユーザー処置: **MQeQueueManager.close** メソッドを使用して、キュー・マネージャーを停止してから、その復元または活動化を試みます。

Except_QMgr_AlreadyExists - 103

説明: キュー・マネージャーが、レジストリーまたは接続定義にすでに存在しています。

ユーザー処置: 新しいキュー・マネージャー名を選択します。

Except_QMgr_InvalidQMgrName - 101

説明: キュー・マネージャー名が命名基準に準拠していません。

ユーザー処置: キュー・マネージャー名は 1 文字以上の ASCII ストリングで、値の範囲は 21 から 127 までです。使用できる文字は以下のとおりです。

- 大文字 A ～ Z
- 小文字 a ～ z
- 数値 0 ～ 9、ピリオド (.)
- 下線 (_)
- パーセント記号 (%)

最初と最後の文字にはピリオド (.) は使えません。

Except_QMgr_InvalidQName - 104

説明: キュー・マネージャー名とキュー名の組み合わせが無効です。

ユーザー処置: 名前が非ヌルであること、長さがゼロでないことを確認します。

Except_QMgr_NotActive - 100

説明: キュー・マネージャーが、この要求を処理するのにアクティブな状態にありません。

ユーザー処置: **MQeQueueManager.activate** メソッドを使用して、キュー・マネージャーを始動し、MQeQueueManager.close メソッドを使用して、キュー・マネージャーを停止します。キュー・マネージャーを再始動して、要求を繰り返します。

Except_QMgr_QExists - 105

説明: このキュー・マネージャーとキューの組み合わせでは、キュー、キュー別名、ホーム・サーバー・キュー、またはストア・アンド・フォワード・キュー定義が、すでに存在しています。

ユーザー処置: 指定の宛先に対しては、ホーム・サーバー・キューまたはストア・アンド・フォワード・キュー定義が、 1 つしか許可されません。

Except_QMgr_NotConfigured - 113

説明: キュー・マネージャーのレジストリーが、正しく妥当性検査できません。

ユーザー処置: レジストリーのサブディレクトリーとファイルが、 ini ファイルで指定した正しい場所にあるかどうかを確認します。

Except_QMgr_QDoesNotExist - 108

説明: このキュー・マネージャーとキューの組み合わせでは、キュー、キュー別名、またはストア・アンド・フォワード・キュー定義が存在しません。

ユーザー処置: キューまたはキュー別名を作成するか、ストア・アンド・フォワード・キューにキュー・マネージャーを追加します。

Except_QMgr_QNotEmpty - 107

説明: まだメッセージが入っているキューを、削除しようとしました。

ユーザー処置: 削除する前に、このキューを空にする必要があるかどうかを判断します。これが非同期キューである場合もあり、その場合は、そこに入っているメッセージをリモート・キュー・マネージャーに転送することができません。

Except_QMgr_UknownQMgr - 106

説明: 参照されたキュー・マネージャーは、このキュー・マネージャーでは定義されていません。

ユーザー処置: リモート・キュー・マネージャーに対して、定義を作成します。

Except_QMgr_WrongQtype - 110

説明: 作成できるキューは、タイプが MQeQueue.Queue_Synchronous または MQeQueue.Queue_ASynchronous のものだけです。指定されたタイプは認識されません。

ユーザー処置: キュー・タイプの正しい値を指定します。

Except_Reg_AddFailed - 309

説明: 新規レジストリー項目の作成に失敗しました。

ユーザー処置: この例外と共に表示されるテキストに、詳しい説明があります。

Except_Reg_AlreadyExists - 302

説明: ターゲット名がすでに存在するレジストリー項目を、作成または名前変更しようとしています。

ユーザー処置: 新しいターゲット名を選択します。

Except_Reg_AlreadyOpen - 320

説明: レジストリーを 2 度目に活動化しようとしています。

ユーザー処置: レジストリーはすでにオープンしています。もう一度オープンしようとした理由を突き止めます。

Except_Reg_CRTKeyDecFailed - 317

説明: PrivateRegistry クリデンシャルを使用したデータの暗号化解除に、失敗しました。

ユーザー処置: この例外と共に表示されるテキストに、詳しい説明があります。

Except_Reg_CRTKeySignFailed - 318

説明: PrivateRegistry クリデンシャルを使用した、データへのディジタル署名 (ISO9796) に失敗しました。

ユーザー処置: この例外と共に表示されるテキストに、詳しい説明があります。

Except_Reg_DeleteFailed - 310

説明: レジストリー項目の削除に失敗しました。

ユーザー処置: この例外と共に表示されるテキストに、詳しい説明があります。

Except_Reg_DeleteRegistryFailed - 319

説明: レジストリーのクリーンアップに失敗しました。

ユーザー処置: この例外と共に表示されるテキストに、詳しい説明があります。

Except_Reg_DoesNotExist - 303

説明: レジストリー項目が見つかりません。

ユーザー処置: 項目が存在しない理由を突き止めます。

Except_Reg_InvalidSession - 305

説明: レジストリー・セッションがアクティブでないか、 LocalRegType が指定されていません。

ユーザー処置: LocalRegType 項目が、キュー・マネージャー・スタートアップ ini ファイルの、 `[Registry]` セクションにあることを確認します。

Except_Reg_ListFailed - 313

説明: レジストリー項目のリスト表示に失敗しました。

ユーザー処置: この例外と共に表示されるテキストに、詳しい説明があります。

Except_Reg_NotDefined - 306

説明: レジストリー・タイプがヌルです。

ユーザー処置: レジストリー・タイプを、
QueueManager、Queue、RemoteQMgr のいずれかにします。

Except_Reg_NotSecure - 321

説明: セキュア・レジストリーが必要ですが、使用できません。

ユーザー処置: レジストリーがセキュアでない理由を突き止めます。

Except_Reg_NullName - 301

説明: 引き数の 1 つがヌルであるため、レジストリー項目の名前変更に失敗しました。

ユーザー処置: 引き数がヌルになっている理由を突き止めて、訂正します。

Except_Reg_OpenFailed - 304

説明: レジストリーがオープンできません。

ユーザー処置: この例外と共に表示されるテキストに、詳しい説明があります。ファイルを検査して、アクセス権を訂正します。

Except_Reg_ReadFailed - 311

説明: レジストリー項目の読み取りに失敗しました。

ユーザー処置: この例外と共に表示されるテキストに、詳しい説明があります。

Except_Reg_RenameFailed - 315

説明: レジストリー項目の名前変更に失敗しました。

ユーザー処置: この例外と共に表示されるテキストに、詳しい説明があります。

Except_Reg_ResetPINFailed - 316

説明: レジストリー項目での PIN のリセットに失敗しました。

ユーザー処置: この例外と共に表示されるテキストに、詳しい説明があります。

Except_Reg_SearchFailed - 314

説明: レジストリー項目の検索に失敗しました。

ユーザー処置: この例外と共に表示されるテキストに、詳しい説明があります。

Except_Reg_UpdateFailed - 312

説明: レジストリー項目の更新に失敗しました。

ユーザー処置: この例外と共に表示されるテキストに、詳しい説明があります。

Except_Registry_Update_Failed - 605

説明: 特定のブリッジ・リソースの特性の値を作成または更新するために、管理メッセージが MQSeries-Bridge に送信されました。ブリッジは、新しく更新された値をレジストリーに保管しようとしましたが、保管操作は失敗しました。

ユーザー処置: レジストリーが、情報を保管できなかった理由を調べて、操作を再試行します。

Except_Resource_Not_Found - 614

説明: 管理メッセージが、存在していない MQSeries-Bridge リソースに送信されました。(たとえば、存在していないリスナーに停止メッセージを送信しました)。

ユーザー処置: 管理メッセージの「名前 (name)」フィールドをすべて調べて、そのすべてにオブジェクトが存在することを確認します。

Except_Rule - 013

説明: MQeQueueRules クラスがロードできません。あるいは、MQSeries Everyplace ルール・クラスが、このアクションを許可すべきでないと判断しました。

ユーザー処置:

Except_S_BadIntegrity - 507

説明: MQeMAttribute、または MQeMTrustAttribute で保護されているメッセージの一部として送信されたデータ・ダイジェストが、メッセージのデータと対応しません。メッセージが改ざんされた可能性があります。

ユーザー処置: セキュリティー管理者に、問題を報告してください。

Except_S_BadSubject - 510

説明: 最小限の証明でエンコードされたサブジェクト名が、レジストリーの証明書の名前と一致しません。証明書が改ざんされた可能性があります。

ユーザー処置: セキュリティー管理者に、問題を報告してください。

Except_S_Cipher - 501

説明: データの暗号化解除に、誤った暗号または鍵が使用されました。

ユーザー処置: 正しい暗号および鍵が指定されていることを確認します。

Except_S_CertificateExpired - 503

説明: 最小限の証明が有効期限切れです。

ユーザー処置: セキュリティー管理者に問題を報告して、証明書を更新してください。

Except_S_MiniCertNotAvailable - 505

説明: 最小限の証明が使用できません。この例外と共に表示されるテキストで、検出できなかった証明書が指摘されています。

ユーザー処置: これによって、 MQeMTrustAttribute によるメッセージの書き込みまたは取得が発生した場合は、送信側および受信側の最小限の証明が、適切な公開レジストリーで使用できることを確認してください。これが MQeWTLSCertAuthenticator の使用によって発生した場合は、参加しているキュー・マネージャーまたはキューのいずれかが、そのクリデンシャルを正しい方法で入手しなかった可能性があります。

セキュリティー管理者に、問題を報告してください。

Except_S_InvalidAttribute - 504

説明: この例外が throw される理由はいくつか考えられます。

ユーザー処置: この例外と共に表示されるテキストに、詳しい説明があります。

Except_S_InvalidSignature - 502

説明: ディジタル署名が ISO9796 の検証に失敗しました。この例外と共に表示されるテキストが ″validating data ...″ で始まっている場合は、MQeMTrustAttribute を使用して送信されたメッセージの署名を検査したときに、この例外が発生しました。この例外と共に表示されるテキストが ″MiniCert = ″ で始まっている場合は、テキストで指定されている最小限の証明が無効です。

ユーザー処置: セキュリティー管理者に、問題を報告してください。

Except_S_MissingSection - 509

説明: 最小限の証明サーバーのスタートアップ・パラメーターから、セクションが欠落しています。

ユーザー処置: すべてのセクションが、スタートアップ・パラメーターに組み込まれていることを確認します。

Except_S_NoPresetKeyAvailable - 508

説明: MQeMAttribute を使用して、メッセージの書き込みまたは送信を行う際に、鍵が提供されていません。

ユーザー処置: 鍵のある属性を提供します。

Except_S_RegistryNotAvailable - 506

説明: 専用レジストリーまたは公開レジストリーが使用できません。

ユーザー処置: これによって、 MQeMTrustAttribute によるメッセージの書き込みまたは取得が発生した場合は、属性に正しい専用レジストリーおよび公開レジストリーを割り当てていることを確認してください。これが MQeWTLSCertAuthenticator の使用によって発生した場合は、参加しているキュー・マネージャーまたはキューのいずれかが、正しくセットアップされていない可能性があります。

セキュリティー管理者に、問題を報告してください。

Except_ShutDown_Children_First - 626

説明: 予期しないエラーが発生しました。

ユーザー処置: お手近のサポート担当者に連絡してください。

Except_Stopped - 009

説明: 通信チャネルの状態が正しくありません。この状態は MQSeries Everyplace が内部で処理するはずなので、問題にはなりません。

ユーザー処置: ルール・クラスを拡張したり、 MQSeries Everyplace のピアツーピア・ソリューションを開発したりする場合に、この例外を処理しなければならないことがあります。

Except_Syntax - 003

説明: この例外が throw されるのは、 MQSeries Everyplace オブジェクトが特定のフォーマットまたはスタイルのデータを予期しているのに、データのフォーマットまたはスタイルが異なる場合です。

ユーザー処置: この例外と共に表示されるテキストで、問題の本質に関する手掛かりが得られるはずです。たとえば、MQSeries Everyplace アダプターには、ストリング中に少なくとも 1 つのコロン (:) を含む (`Network:127.0.0.1:8080` のように)、正しい仕様が必要です。

Except_Timeout - 014

説明: この例外が throw されるのは、 MQSeries Everyplace チャネルがタイムアウト期間後の接続に失敗した場合です。

ユーザー処置: リモート・キュー・マネージャーが、コンタクト可能であることを確認します。ネットワーキング問題が、この状態の原因である可能性があります。

Except_Transformation_Error - 615

説明: MQSeries メッセージを、 MQSeries Everyplace メッセージに変換しようとしたときに (あるいは、 MQSeries Everyplace メッセージを MQSeries メッセージに変換しようとしたときに)、問題が発生しました。

ユーザー処置: カスタム・トランスフォーマーを使用している場合は、アプリケーション・ロジックを調べて、メッセージが正しく変換されるようにしてください。

Except_Transport_Request - 041

説明: 未知の MQSeries Everyplace コマンド要求を受信しました。

ユーザー処置: お手近の IBM サポート担当者に連絡してください。

Except_Type - 004

説明: MQSeries Everyplace メソッドに渡されたクラス・オブジェクトのタイプが、正しくありません。

ユーザー処置: この例外と共に表示されるテキストで、この問題を解決するための追加情報が得られるはずです。

第3部 付録

付録. 特記事項

本書は米国 IBM 社が提供する製品およびサービスについて作成したものであり、米国以外の国においては本書で述べる製品、サービス、または機能を提供しない場合があります。日本で利用可能な製品、サービス、およびフィーチャーについては、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらのプログラムまたは製品に代えて、IBM の知的所有権を侵害することのない機能的に同等のプログラムまたは製品を使用することができます。ただし、IBM 製以外の製品と組み合わせた場合、その操作の評価と検証については、お客様の責任で行っていただきます。

IBM は、本書で解説されている主題について特許権 (特許出願を含む)、商標権、または著作権を所有している場合があります。本書の提供は、これらの特許権、商標権、および著作権について、本書で明示されている場合を除き、実施権、使用権等を許諾することを意味するものではありません。実施権、使用権等の許諾については、下記の宛先に、書面にてご照会ください。

〒106-0032 東京都港区六本木 3 丁目 2-31
AP 事業所
IBM World Trade Asia Corporation
Intellectual Property Law & Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。
IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

本書に対して、周期的に変更が行われ、これらの変更は、文書の次版に組み込まれます。 IBM は、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、 (i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire
England
SO21 2JN

本プログラムに関する上記の情報は、適切な条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

著作権:

本書には、さまざまなオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。これらの例は、すべての場合について完全にテストされたものではありません。 IBM はこれらのプログラムの信頼性、可用性、および機能について法律上の瑕疵担保責任を含むいかなる明示または暗示の保証責任も負いません。

商標

次のものは、IBM Corporation の米国およびその他の国における商標です。

IBM
MQSeries

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

Windows NT は Microsoft Corporation の米国およびその他の国における商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標または登録商標です。

参照文献

関連資料:

- *MQSeries Everyplace for Multiplatforms 紹介*, (GC88-8653-00)
- *MQSeries Everyplace for Multiplatforms プログラミング・ガイド*, (SC88-8654-00)
- *MQSeries An Introduction to Messaging and Queuing*, (GC33-0805-01)
- *MQSeries (Windows NT 版) インストール・ガイド V5.1*, (GD88-7162-00)

Printed in Japan

SC88-8655-02

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12