Fiorano
Enabling change at the speed of thought

www.fiorano.com

FioranoMQ® 9

C++ RTL Native Guide

AMERICA'S
Fiorano Software, Inc.
718 University Avenue Suite 212, Los Gatos,
CA 95032 USA
Tel: +1 408 354 3210
Fax: +1 408 354 0846
Toll-Free: +1 800 663 3621
Email: info@fiorano.com

EMEA
Fiorano Software Ltd.
3000 Hillswood Drive Hillswood Business Park Chertsey Surrey
KT16 0RS UK
Tel: +44 (0) 1932 895005
Fax: +44 (0) 1932 325413
Email: info_uk@fiorano.com

APAC
Fiorano Software Pte. Ltd.
Level 42, Suntec Tower Three 8
Temasek Boulevard 038988
Singapore
Tel: +65 68292234
Fax: +65 68292235
Email: info_asiapac@fiorano.com

Fiorano

Content

Chapter 1: Introduction

The Native C++Runtime Library (C++RTL) allows C++ based application to interact with FioranoMQ. A C++ based client can thus seamlessly communicate with Java based Fiorano Clients.

This version of C++RTL is designed to run on both native and .NET platforms. Both the versions support secure and non-secure TCP and HTTP connections on Win32 platform for Point-to-Point and Publish/Subscribe communication models.

The C++ Runtime is designed to provide maximum conformance with the JMS specifications. All public APIs have similar signature as the corresponding java APIs specified by JMS. The classes have similar naming convention.

This guide shows you how to use all the ActiveX Runtime Library functions, to create Tifosi Services in VB and Delphi. This is a reference guide discussing all APIs of Tifosi ActiveX RTL in detail, assuming you know sufficient about VB and Delphi to read and understand the sample code.

The ActiveX Runtime Service APIs allow inter-operability between services, and provide the same functionality as the Java Services Development APIs.

The Native C++Runtime Library (C++RTL) allows C++ based application to interact with FioranoMQ. A C++ based client can thus seamlessly communicate with Java based Fiorano Clients.

This version of C++RTL is designed to run on both native and .NET platforms. Both the versions support secure and non-secure TCP and HTTP connections on Win32 platform for Point-to-Point and Publish/Subscribe communication models.

The C++ Runtime is designed to provide maximum conformance with the JMS specifications. All public APIs have similar signature as the corresponding java APIs specified by JMS. The classes have similar naming convention.

Chapter 2: Datatypes and Constants

This chapter contains an overview of the C++RTL specific data types and constants. A brief explanation of each data type along with sizes, is provided.

Basic Data Types and their Sizes

This section lists the basic data types used in C++RTL APIs, indicating the size of each.

- mqbyte Data defined to occupy 8 bits (unsigned).
- mqchar Data defined to occupy 8 bits.
- mqshort Data defined to occupy 16 bits.
- mqint Data defined to occupy 32 bits.
- mqlong Data defined to occupy 64 bits.
- mqfloat Data defined to occupy 32 bits.
- mqdouble Data defined to occupy 64 bits.

For more information on the data types, refer to section data types and constants of FioranoMQ C++RTL guide.

C++RTL Constants

All required public constants are categorized and declared in the related classes according to JMS specifications.

Class	Constant

Cmessage	`static const int CDEFAULT_DELIVERY_MODE`
	`static const int CDEFAULT_PRIORITY`
	`static const int CDEFAULT_TIME_TO_LIVE`

CdeliveryMode	`static const int CNON_PERSISTENT;` `static const int CPERSISTENT SET_CPERSISTENT;`

Class	Constant

CJMSSession	`static const int CAUTO_ACKNOWLEDGE`
	`static const int CCLIENT_ACKNOWLEDGE`
	`static const int CDUPS_OK_ACKNOWLEDGE`
	`static const int CSESSION_TRANSACTED`

For more information on these constants, refer to the Java docs of the containing class.

Naming convention

The C++RTL adheres to the following naming convention for you to easily identify the classes, constants and member functions with the corresponding definitions in JMS specifications.

Type	Naming Convention	Example
Class	C<JMSClass name>	`CTopicPublisher(JMS:TopicPub-lisher)`
Constant	C<JMS constants name>	`CNON_PERSISTENT()`
Function	same as in JMS spec	`publish`

This guide shows you how to use all the ActiveX Runtime Library functions, to create Tifosi Services in VB and Delphi. This is a reference guide discussing all APIs of Tifosi ActiveX RTL in detail, assuming you know sufficient about VB and Delphi to read and understand the sample code.

The ActiveX Runtime Service APIs allow inter-operability between services, and provide the same functionality as the Java Services Development APIs.

Chapter 3: Error Handling

The C++RTL uses exceptions to provide error handling. In event of an error in the C++RTL layer, CJMSException with a specific errorCode and description is thrown.

The Exception can be caught at the application level and the associated message can be read using the public API getMessage(). The exception handling is also exhaustive in that it provides the complete function stack trace at the moment of the error.

The exception stack is maintained in the Thread Local Storage, so that the excep- tion that occurred in one thread doesn't interfere with the flow of other threads. The stack trace can be printed on the console using the API call printStackTrace().

```
try
{
//Application code
}
catch(CJMSException *e)
                              {
                              cout << e->getMessage();
                              e->printStackTrace();
                              delete e;
                              }
```

For more information, read the CJMSException**Error! Reference source not found.** section.

Chapter 4: The DotNet Version

Microsoft's .NET™ is one of the emerging standards that is changing the way enterprises build Application software. The .NET technologies enable greater agility and increased flexibility for integrating business processes within internal organizations or with external business partners.

Fiorano's MQ provides support for .NET platform through the .NET version of C++ RTL. Any .NET application can be compiled with this rtl and deployed as a dotnet component. Such a component can communicate with any other component, written in any supported language and executing on any supported windows/non-windows platform.

The key point to note here is that these .NET applications execute in their native form without the need for conversion into web services. This is a unique feature that leads to high performance and lower maintenance costs and makes it ahead of all other EAI/BPM platforms in terms of interoperability.

The DotNet version of the native CppRTL has been implemented using the managed extensions provided with Microsoft Visual C++ ™. The dotNet version provides all the functionality of CppRTL with additional support for garbage collection provided by the dotnet platform.

Installation

The same set of header files can be used while compiling dotnet applications with FioranoMQ.

Make sure to compile with the DOTNET preprocessor definition. Follow the same set of instructions listed above.

Using DotNet samples

Organization of samples

The native C++ samples are located in the cpp\native\samples\ folder in FioranoMQ installation and organized under the following heads:

PubSub This directory contains the following sample programs, which illustrate basic JMS Publish/Subscribe functionality, using the C++ RTL.

PTP This directory contains two sample programs which illustrate JMS Request-Reply mechanism using the CppRTL.

Compiling the samples

To run the samples using FioranoMQ,

Compile each of the source files using the script files in the fmq\clients\cpp\native\scripts folder.

The scripts directory contains a script cppclientbuild.bat which compiles the samples using the Microsoft VC++ compiler.

Running the samples

Refer to the readme.txt file in the specific directory for information(if any) on the runtime arguments for the executable.

Chapter 5: API Reference

This document lists all FioranoMQ C++ Runtime APIs by category. The JMSInterface APIs are those that are common for both Publish/subscribe and PTP semantics, Naming and JNDI APIs are those that perform naming operations, Helper Function APIs are the utility functions, Publish/Subscribe APIs are those that implement the JMS Publish/Subscribe semantics and PTP APIs are those that implement the JMS Point To Point semantics.

The organization of this document is summarized as follows.

Helpers

CHashTable

This class implements a hashtable, which maps keys to values. Any non-null mqobject can be used as a value and any non-null mqcstring can be used as a key.

Constructor

Constructs a new, empty hashtable.

Declaration

```
CHashTable()
            throw (CJMSException *);
```

Put

Maps the specified key to the specified value in this hashtable. Neither the key nor the value can be null.

Declaration

```
mqobject Put(mqcstring key, mqobject value)

               throw (CJMSException *);
```

Get

Returns the value to which the specified key is mapped in this hashtable.

Declaration

```
mqobject Get(mqcstring key) FMQCONST

              throw (CJMSException *);
```

RemoveElement

Removes the key (and its corresponding value) from this hashtable. This method does nothing if the key is not in the hashtable.

Declaration

```
mqobject RemoveElement(mqcstring key)
                            throw (CJMSException *);
```

HashTableEnumerator

CHashTableEnumerator generates a series of elements, one at a time. Successive calls to the nextElement method return successive elements of the series.

hasMoreElemets

Tests if this enumeration contains more elements. Returns true if enumeration contains more elements otherwise it returns false

Declaration

```
mqboolean hasMoreElements();
```

nextKeyElement

Returns the nextKey value of this enumeration if this enumeration object has at least one more element to provide.

Declaration

```
mqcstring nextKeyElement();
```

Returns

Returns the next Key value

nextValueElement

Returns the next value element of this enumeration if thisenumeration object has at least one more element to provide

Declaration

```
mqobject nextValueElement();
```

Returns

Returns the next value Element

JMS Interfaces

CAdvisoryMessage

Base class for all advisory messages.

getAdvisoryMsgString

Returns a Unicode string for the advisory message.

Declaration

```
mqcstring_unicode getAdvisoryMsgString()
throw (CJMSException *);
```

Returns

The Unicode string value for the advisory message string

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

getAMState

Returns a mqint object representing the Advisroy message state

Declaration

```
mqint getAMState()
throw (CJMSException *);
```

Returns

The advisory message state as mqint object

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

isActive

Returns a mqboolean value representing the status of the advisory message object.

Declaration

```
mqboolean isActive()
throw (CJMSException *);
```

Returns

Returns True if advisory message object is active, FALSE otherwise.

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

isDisconnected

Returns a mqboolean value representing the connection status of the advisory message object.

Declaration

```
mqboolean isDisconnected()
throw (CJMSException *);
```

Returns

Returns True if advisory message object is disconnected, FALSE otherwise.

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

isRevalidating

Returns mqboolean value representing the revalidation status of the advisory message object.

Declaration

```
mqboolean isRevalidating()
throw (CJMSException *);
```

Returns

Returns TRUE if revalidating thread is active, FALSE otherwise.

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

isFailed

Returns mqboolean value representing the failure staus of advisory message thread.

Declaration

```
mqboolean isFailed()
throw (CJMSException *);
```

Returns

Returns TRUE if is failed, FALSE otherwise.

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

isTransferring

Returns mqboolean value if the advisory message thread is transferring data.

Declaration

```
mqboolean isTransferring()
throw (CJMSException *);
```

Returns

Returns TRUE if is transferring, FALSE otherwise.

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

isTransferComplete

Returns mqboolean value representing the transfer status.

Declaration

```
mqboolean isTransferComplete()
throw (CJMSException *);
```

Returns

Returns TRUE if the transfer is complete, FALSE otherwise.

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

CAdvisoryMsgListener

A CAdvisoryMsgListener object is used to receive asynchronously delivered advisory messages when the reconnection thread is active.

onAdvisoryMessage

Passes a message to the listener.

Declaration

```
virtual void onAdvisoryMessage(CAdvisoryMessage *msg) = 0;
```

Parameters

msg

The message passed to the listener

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

CByteMessage

A CBytesMessage object is used to send a message containing a stream of uninterpreted bytes. It inherits from the CMessageinterface and adds a bytes message body. The receiver of the message supplies the interpretation of the bytes.

Derives

CMessage

getBodyLength

Gets the number of bytes of the message body when the message is in read-only mode. The value returned can be used to allocate a byte array. The value returned is the entire length of the message body, regardless of where the pointer for reading the message is currently located.

Declaration

```
mqlong getBodyLength() FMQCONST
throw (CJMSException  *);
```

Returns

Number of bytes in the message.

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

readBoolean

Reads a boolean from the bytes message stream.

Declaration

```
mqboolean readBoolean() FMQCONST
throw (CJMSException  *);
```

Returns

The boolean value read

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

readByte

Reads a signed 8-bit value from the bytes message stream.

Declaration

```
mqbyte readByte() FMQCONST
throw (CJMSException  *);
```

Returns

The next byte from the bytes message stream as a signed 8-bit byte

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

readBytes

Reads a byte array from the bytes message stream.If the length of array value is less than the number of bytes remaining to be read from the stream, the array should be filled. A subsequent call reads the next increment, and so on. If the number of bytes remaining in the stream is less than the length of array value, the bytes should be read into the array. The return value of the total number of bytes read will be less than the length of the array, indicating that there are no more bytes left to be read from the stream. The next read of the stream returns -1.

Declaration

```
mqint readBytes(mqbyteArray value, int length) FMQCONST
throw (CJMSException  *);
```

Parameters

value

The buffer into which the data is read.

Length

The length of the buffer.

Returns

The total number of bytes read into the buffer, or -1 if there is no more data because the end of the stream has been reached.

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

readChar

Reads a Unicode character value from the bytes message stream.

Declaration

```
mqchar readChar() FMQCONST
 throw (CJMSException  *);
```

Returns

The next two bytes from the bytes message stream as a Unicode character

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

readDouble

Reads a double from the bytes message stream.

Declaration

```
mqdouble readDouble() FMQCONST
                            throw (CJMSException  *);
```

Returns

The next eight bytes from the bytes message stream, interpreted as a double.

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

readFloat

Reads a float from the bytes message stream.

Declaration

```
mqfloat readFloat() FMQCONST
                              throw (CJMSException  *);
```

Returns

The next four bytes from the bytes message stream, interpreted as a float

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

readInt

Reads a signed 32-bit integer from the bytes message stream.

Declaration

```
mqint readInt() FMQCONST
                              throw (CJMSException  *);
```

Returns

The next four bytes from the bytes message stream, interpreted as an int

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

readLong

Reads a signed 64-bit integer from the bytes message stream.

Declaration

```
mqlong readLong() FMQCONST
                              throw (CJMSException  *);
```

Returns

The next eight bytes from the bytes message stream, interpreted as a long.

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

readShort

Reads a signed 16-bit number from the bytes message stream.

Declaration

```
mqshort readShort() FMQCONST
                            throw (CJMSException  *);
```

Returns

The next two bytes from the bytes message stream, interpreted as a signed 16-bit number

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

readUnsignedByte

Reads an unsigned 8-bit number from the bytes message stream.

Declaration

```
mqint readUnsignedByte() FMQCONST
                            throw (CJMSException  *);
```

Returns

The next byte from the bytes message stream, interpreted as an unsigned 8-bit number

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

readUnsignedShort

Reads an unsigned 16-bit number from the bytes message stream.

Declaration

```
mqint readUnsignedShort() FMQCONST
                            throw (CJMSException  *);
```

Returns

The next two bytes from the bytes message stream, interpreted as an unsigned 16-bit integer.

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

readUTF

Reads a mqcstring that has been encoded using a modified UTF-8 format from the bytesmessage stream.

For more information on the UTF-8 format, see "File System Safe UCS Transformation Format (FSS_UTF)", X/Open Preliminary Specification, X/Open Company Ltd., Document Number: P316. This information also appears in ISO/IEC 10646, Annex P.

Declaration

```
mqcstring readUTF() FMQCONST
                            throw (CJMSException  *);
```

Returns

A Unicode mqcstring from the bytes message stream

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

reset

Puts the message body in read-only mode and repositions the stream of bytes to the beginning.

Declaration

```
void reset()
                            throw (CJMSException  *);
```

Throws

CJMSException

If the JMS provider fails to reset the message due to some internal error.

writeBoolean

Writes a boolean to the bytes message stream as a 1-byte value. The value true is written as the value (byte)1; the value false is written as the value (byte)0.

Declaration

```
void writeBoolean(mqboolean value)
                            throw (CJMSException  *);
```

Parameters

value

The boolean value to be written

Throws

CJMSException

If the JMS provider fails to write the message due to some

internal error.

writeByte

Writes a byte to the bytes message stream as a 1-byte value.

Declaration

```
void writeByte(mqbyte value)
                            throw (CJMSException  *);
```

Parameters:

value

The byte value to be written

Throws

CJMSException

If the JMS provider fails to write the message due to some internal error.

writeBytes

Writes a portion of a byte array to the bytes message stream.

Declaration

```
void writeBytes(mqbyteArray value, mqint offset, mqint length)
                            throw (CJMSException  *);
```

Parameters

value

The byte array value to be written
offset

The initial offset within the byte array
length

The number of bytes to use

Throws

CJMSException

If the JMS provider fails to write the message due to some internal error.

writeChar

Writes a char to the bytes message stream as a 2-byte value, high byte first.

Declaration

```
void writeChar(mqchar value)
                            throw (CJMSException  *);
```

Parameters

value

The char value to be written

Throws

CJMSException

If the JMS provider fails to write the message due to some internal error.

writeDouble

Reads a double from the bytes message stream.

Declaration

```
void writeDouble(mqdouble value)
                            throw (CJMSException  *);
```

Returns

The next eight bytes from the bytes message stream, interpreted as a double

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

writeFloat

Converts the float argument to an int using the floatToIntBits method in class Float, and then writes that int value to the bytes message stream as a 4-byte quantity, high byte first.

Declaration

```
void writeFloat(mqfloat value)
                            throw (CJMSException  *);
```

Parameters

value

The float value to be written

Throws

CJMSException

If the JMS provider fails to write the message due to some internal error.

writeInt

Writes an int to the bytes message stream as four bytes, high byte first.

Declaration

```
void writeInt(mqint value)
                            throw (CJMSException  *);
```

Parameters

value

The int to be written

Throws

CJMSException

If the JMS provider fails to write the message due to some internal error.

writeLong

Writes a long to the bytes message stream as eight bytes, high byte first.

Declaration

```
void writeLong(mqlong value)
throw (CJMSException  *);
```

Parameters

value

The long to be written

Throws

CJMSException

If the JMS provider fails to write the message due to some internal error.

writeShort

Writes a short to the bytes message stream as two bytes, high byte first.

Declaration

```
void writeShort(mqshort value)
                            throw (CJMSException  *);
```

Parameters

value

The short to be written

Throws

CJMSException

If the JMS provider fails to write the message due to some internal error.

writeUTF

Writes a mqcstring to the bytes message stream using UTF-8 encoding in a machine-independent manner.

For more information on the UTF-8 format, refer "File System Safe UCS Transformation Format (FSS_UTF)", X/Open Preliminary Specification, X/OpenCompany Ltd., Document Number: P316. This information also appears in ISO/IEC 10646, Annexure P.

Declaration

```
void writeUTF(mqcstring value)
                            throw (CJMSException  *);
```

Parameters

value

The String value to be written

Throws

CJMSException

If the JMS provider fails to write the message due to some internal error.

CConnection

A CConnection object is a client's active connection to its JMS provider.

Base for

CQueueConnection CTopicConnection

CConnectionConsumer

For application servers, CConnection objects provide a special facility for creating a ConnectionConsumer.

CConnectionFactory

A CConnectionFactory object encapsulates a set of connection configuration parameters that has been defined by an administrator. A client uses it to create a connection with a JMS provider.

Base for

CTopicConnectionFactory CQueueConnectionFactory

CDestination

A CDestination object encapsulates a provider-specific address.

Base for

Ctopic CQueue CTemporaryTopic CTemporaryQueue

CFioranoException

This is the root class of all exceptions.

public

checkForException

Utility Function that throws CJMSException if an exception had occurred.

Declaration

```
static void checkForException()
                            throw (CJMSException *);
```

CJMSException

Constructor

Constructs a CJMSException with the specified reason and with the error code.

Declaration

```
CJMSException(mqcstring errCode);
```

Parameters

errCode

The error code that identifies the error

Constructor

Constructs a CJMSException with the specified reason and with the error code and description.

Declaration

```
CJMSException(mqcstring errCode, mqcstring errDesc);
```

Parameters

errCode

The error code that identifies the error
errDesc

description of the error

checkForException

Utility Function that throws CJMSException if an exception had occurred.

Declaration

```
static void static void checkForException(mqcstring errCode);
                        throw (CJMSException *);
```

Parameters

errCode

The error code that identifies the error

Another over-loaded function

Utility Function that throws CJMSException if an exception had occurred.

Declaration

```
static void checkForException(mqcstring errCode, mqcstring errDesc)
                        throw (CJMSException *);
```

Parameters

errCode

The error code that identifies the error
errDesc

of the error

printStackTrace

Prints the Function stack trace on the console

Declaration

```
void printStackTrace();
```

getStackTrace

Gets the stack trace of the function

Declaration

```
const DN_STRING getStackTrace();
```

getErrorCode

Gets the vendor-specific error code.

Declaration

```
const mqcstring getErrorCode();
```

Returns

The vendor-specific error code

getLinkedException

Gets the exception linked to this one

Declaration

```
CJMSException *getLinkedException();
```

Returns

The linked exception

CExceptionListener

If FioranoMQ cpp detects a serious problem with a CConnection object, it informs the CConnection object's CExceptionListener, if one has been registered. It does this by calling the listener's onException method, passing it a CJMSException argument describing the problem.

onException

Notifies the user of a JMS exception. The user is expected to override this function with the required functionality

Declaration

```
virtual void onException(CFioranoException *msg) = 0;
```

Parameters

msg

Message handle

CMapMessage

The CMapMessage object is used to send a set of name-value pairs. The names must have a value that is not null, and not an empty mqcstring. The entries can be accessed sequentially or randomly by name. CMapMessage inherits from the CMessage interface and adds a message body that contains a Map.

Derives

CMessage

getBoolean

Returns the boolean value with the specified name.

Declaration

```
mqboolean getBoolean(mqcstring name) FMQCONST
                            throw (CJMSException *);
```

Parameters

name

The name of the boolean

Returns

The boolean value with the specified name

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

getByte

Returns the byte value with the specified name.

Declaration

```
mqbyte getByte(mqcstring name) FMQCONST
                              throw (CJMSException *);
```

Parameters

name

The name of the byte

Returns

The byte value with the specified name.

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

getChar

Returns the Unicode character value with the specified name.

Declaration

```
mqchar getChar(mqcstring name) FMQCONST
                              throw (CJMSException *);
```

Parameters

name

The name of the Unicode character

Returns

The Unicode character value with the specified name

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

getDouble

Returns the double value with the specified name.

Declaration

```
mqdouble getDouble(mqcstring name) FMQCONST
                            throw (CJMSException *);
```

Parameters

name

The name of the double

Returns

The double value with the specified name

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

getFloat

Returns the float value with the specified name.

Declaration

```
mqfloat getFloat(mqcstring name)FMQCONST
                            throw (CJMSException *);
```

Parameters

name

The name of the float

Returns

The float value with the specified name

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

getInt

Returns the int value with the specified name.

Declaration

```
mqint getInt(mqcstring name) FMQCONST
                            throw (CJMSException *);
```

Parameters

name

The name of the int

Returns

The int value with the specified name

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

getLong

Returns the long value with the specified name.

Declaration

```
mqlong getLong(mqcstring name) FMQCONST
                              throw (CJMSException *);
```

Parameters

name

The name of the long

Returns

The long value with the specified name

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

getMapNames

Returns a pointer to mqcstrings of all the names in the MapMessage object.

Declaration

```
mqcstring *getMapNames() FMQCONST
                              throw (CJMSException *);
```

Returns

An a pointer to all the names (mqcstring) in this MapMessage

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

getMapNamesHTEnum

Returns an Enumeration of all the names in the MapMessage object.

Declaration

```
CHashTableEnumerator *getMapNamesHTEnum() FMQCONST
                            throw (CJMSException *);
```

Returns

An enumeration of all the names in this MapMessage

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

getShort

Returns the short value with the specified name.

Declaration

```
mqshort getShort(mqcstring name) FMQCONST
                            throw (CJMSException *);
```

Parameters

name

The name of the short

Returns

The short value with the specified name

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

getString

Returns the String value with the specified name.

Declaration

```
mqcstring getString(mqcstring name) FMQCONST
                            throws CJMSException;
```

Parameters

name

The name of the String

Returns

The String value with the specified name; if there is no item by this name, a null value is returned

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

itemExists

Indicates whether an item exists in this MapMessage object.

Declaration

```
mqboolean itemExists(mqcstring name) FMQCONST
                            throw (CJMSException *);
```

Parameters

name

The name of the item to test

Returns

TRUE if the item exists

Throws

CJMSException

If the JMS provider fails to determine if the item exists due to some internal error.

setBoolean

Sets a boolean value with the specified name into the Map.

Declaration

```
void setBoolean(mqcstring name, mqboolean value)
                            throw (CJMSException *);
```

Parameters

name

The name of the boolean

value

The boolean value to set in the Map

Throws

CJMSException

If the JMS provider fails to write the message due to some internal error.

setByte

Sets a byte value with the specified name into the Map.

Declaration

```
void setByte(mqcstring name, mqbyte value)
                            throw (CJMSException *);
```

Parameters

name

The name of the byte

value

The byte value to set in the Map

Throws

CJMSException

If the JMS provider fails to write the message due to some internal error.

setBytes

Sets a byte array value with the specified name into the Map.

Declaration

```
void setBytes(mqcstring name, mqbyteArray value)
                            throw (CJMSException *);
```

Parameters

name

The name of the byte array

value

The byte array value to set in the Map; the array is copied so that the value for name will not be altered by future modifications

Throws

CJMSException

If the JMS provider fails to write the message due to some internal error.

setBytes

Sets a portion of the byte array value with the specified name into the Map.

Declaration

```
void setBytes(mqcstring name, mqbyteArray value, mqint offset, mqint length)
                            throw (CJMSException *);
```

Parameters

name

The name of the byte array
value

The byte array value to set in the Map
offset

The initial offset within the byte array
length

The number of bytes to use

Throws

CJMSException

If the JMS provider fails to write the message due to some internal error.

setChar

Sets a Unicode character value with the specified name into the Map.

Declaration

```
void setChar(mqcstring name, mqchar value)
                            throw (CJMSException *);
```

Parameters

name

The name of the Unicode character
value

The Unicode character value to set in the Map

Throws

MSException

If the JMS provider fails to write the message due to some internal error.

setDouble

Sets a double value with the specified name into the Map.

Declaration

```
void setDouble(mqcstring name, mqdouble value)
                              throw (CJMSException *);
```

Parameters

name

The name of the double
value

The double value to set in the Map

Throws

CJMSException

If the JMS provider fails to write the message due to some internal error.

setFloat

Sets a float value with the specified name into the Map.

Declaration

```
void setFloat(mqcstring name, mqfloat value)
                              throw (CJMSException *);
```

Parameters

name

The name of the float
value

The float value to set in the Map

Throws

CJMSException

If the JMS provider fails to write the message due to some internal error.

setInt

Sets an int value with the specified name into the Map.

Declaration

```
void setInt(mqcstring name, mqint value)
                            throw (CJMSException *);
```

Parameters

name

The name of the int
value

The int value to set in the Map

Throws

CJMSException

If the JMS provider fails to write the message due to some internal error.

setLong

Sets a long value with the specified name into the Map.

Declaration

```
void setLong(mqcstring name, mqlong value)
                            throw (CJMSException *);
```

Parameters

name

The name of the long
value

The long value to set in the Map

Throws

CJMSException

If the JMS provider fails to write the message due to some internal error.

setShort

Sets a short value with the specified name into the Map.

Declaration

```
void setShort(mqcstring name, mqshort value)
                            throw (CJMSException *);
```

Parameters

name

The name of the short
value

The short value to set in the Map

Throws

CJMSException

If the JMS provider fails to write the message due to some internal error.

setString

Sets a String value with the specified name into the Map.

Declaration

```
void setString(mqcstring name, mqcstring value)
                            throw (CJMSException *);
```

Parameters

name

The name of the String
value

The String value to set in the Map

Throws

CJMSException

If the JMS provider fails to write the message due to some internal error.

CMessage

The CMessage interface is the root interface of all JMS messages. It defines the message header and the acknowledge method used for all messages.

Base for

CTextMessage CBytesMessage CMapMessage CObjectMessage

and

CStreamMessage

Acknowledge

Acknowledges all consumed messages of the session of this consumed message. All consumed JMS messages support the acknowledge method for use when a client has specified that its JMS session's consumed messages are to be explicitly acknowledged. By invoking acknowledge on a consumed message, a client acknowledges all messages consumed by the session that the message was delivered.

Calls to acknowledge are ignored for both transacted sessions and sessions specified to use implicit acknowledgement modes. A client may individually acknowledge each message as it is consumed, or it may choose to acknowledge messages as an application-defined group (which is done by calling acknowledge on the last received message of the group, thereby acknowledging all messages consumed by the session.) Messages that have been received but not acknowledged may be redelivered.

Declaration

```
void acknowledge()
                        throw(CJMSException *);
```

Throws

CJMSException

If the JMS provider fails to acknowledge the messages due to some internal error.

ClearBody

Clears out the message body. Clearing a message's body does not clear its header values or property entries. If this message body was read-only, calling this method leaves the message body in the same state as an empty body in a newly created message.

Declaration

```
void clearBody()
                        throw (CJMSException *);
```

Throws

CJMSException

If the JMS provider fails to clear the message body due to some internal error.

clearProperties

Clears a message's properties. The message's header fields and body are not cleared.

Declaration

```
void clearProperties()
                            throw (CJMSException *);
```

Throws

CJMSException

If the JMS provider fails to clear the message properties due to some internal error.

getBooleanProperty

Returns the value of the boolean property with the specified name.

Declaration

```
mqboolean getBooleanProperty(mqcstring name) FMQCONST
                            throw (CJMSException *);
```

Parameters

name

The name of the boolean property

Returns

The boolean property value for the specified name

Throws

CJMSException

If the JMS provider fails to get the property value due to some internal error.

getByteProperty

Returns the value of the byte property with the specified name.

Declaration

```
mqbyte getByteProperty(mqcstring name) FMQCONST
                            throw (CJMSException *);
```

Parameters

name

The name of the byte property

Returns

The byte property value for the specified name

Throws

CJMSException

If the JMS provider fails to get the property value due to some internal error.

getDoubleProperty

Returns the value of the String property with the specified name.

Declaration

```
mqdouble getDoubleProperty(mqcstring name) FMQCONST
                            throw (CJMSException *);
```

Parameters

name

The name of the String property

Returns

The String property value for the specified name. If there is no property by this name, a null value is returned

Throws

CJMSException

If the JMS provider fails to get the property value due to some internal error.

getFloatProperty

Returns the value of the double property with the specified name.

Declaration

```
mqfloat getFloatProperty(mqcstring name)FMQCONST
                            throw (CJMSException *);
```

Parameters

name

The name of the double property

Returns

The double property value for the specified name

Throws

CJMSException

If the JMS provider fails to get the property value due to some internal error.

getIntProperty

Description

Returns the value of the int property with the specified name.

Declaration

```
mqint getIntProperty(mqcstring name) FMQCONST
                            throw (CJMSException *);
```

Parameters

name

The name of the int property

Returns

The int property value for the specified name

Throws

CJMSException

If the JMS provider fails to get the property value due to some internal error.

getJMSCorrelationID

Gets the correlation ID for the message. This method is used to return correlation ID values that are either provider-specific message IDs or application-specific String values.

Declaration

```
mqcstring getJMSCorrelationID() FMQCONST
                            throw (CJMSException *);
```

Returns

The correlation ID of a message as a String

Throws

CJMSException

If the JMS provider fails to get the correlation ID due to some internal error.

getJMSCorrelationIDAsBytes

Gets the correlation ID as an array of bytes for the message. The use of a byte[] value for JMSCorrelationID is non-portable.

Returns

The correlation ID of a message as an array of bytes

Throws

JMSException

If the JMS provider fails to get the correlation ID due to some internal error.

Declaration

```
mqcstring getJMSCorrelationIDAsBytes() FMQCONST
                            throw (CJMSException *);
```

getJMSDeliveryMode

Gets the DeliveryMode value specified for this message.

Declaration

```
mqint getJMSDeliveryMode() FMQCONST
                            throw (CJMSException *);
```

Returns

The delivery mode for this message

Throws

CJMSException

If the JMS provider fails to get the delivery mode due to some internal error.

getJMSDestination

Gets the Destination object for this message. The JMSDestination header field contains the destination to which the message is being sent. When a message is sent, this field is ignored. After completion of the send or publish method, the field holds the destination specified by the method. When a message is received, its JMSDestination value must be equivalent to the value assigned when it was sent.

Declaration

```
CDestination *getJMSDestination() FMQCONST
                            throw (CJMSException *);
```

Returns

The destination of this message

Throws

CJMSException

If the JMS provider fails to get the destination due tosome internal error.

getJMSExpiration

Gets the message's expiration value. When a message is sent, the JMSExpiration header field is left unassigned. After completion of the send or publish method, it holds the expiration time of the message. This is the sum of the time-to-live value specified by the client and the GMT at the time of the send or publish. If the time-to-live is specified as zero, JMSExpiration is set to zero to indicate that the message does not expire. When a message's expiration time is reached, a provider should discard it.The JMS API does not define any form of notification of message expiration. Clients should not receive messages that have expired; however, the JMS API does not guarantee that this will not happen.

Declaration

```
mqlong getJMSExpiration() FMQCONST
                              throw (CJMSException *);
```

Returns

The time the message expires, which is the sum of the time-to-live value specified by the client and the GMT at the time of the send

Throws

CJMSException

If the JMS provider fails to get the message expiration due to some internal error.

getJMSMessageID

Gets the message ID. The JMSMessageID header field contains a value that uniquely identifies each message sent by a provider. When a message is sent, JMSMessageID can be ignored. When the send or publish method returns, it contains a provider-assigned value. A JMSMessageID is a String value that should function as a unique key for identifying messages in a historical repository. The exact scope of uniqueness is provider-defined. It should at least cover all messages for a specific installation of a provider, where an installation is some connected set of message routers. All JMSMessageID values must start with the prefix 'ID:'. Uniqueness of message ID values across different providers is not required. Since message IDs take some effort to create and increase a message's size, some JMS providers may be able to optimize message overhead if they are given a hint that the messageID is not used by an application. By calling the MessageProducer. setDisableMessageID method, a JMS client enables this potential optimization for all messages sent by that message producer. If the JMS provider accepts this hint, these messages must have the message ID set to null. If the provider ignores the hint, the message ID must be set to its normal unique value.

Declaration

```
mqcstring getJMSMessageID() FMQCONST
                              throw (CJMSException *);
```

Returns

The message ID

Throws

CJMSException

If the JMS provider fails to get the message ID due to someinternal error.

getJMSPriority

Gets the message priority level. The JMS API defines ten levels of priority value, with 0 as the lowest priority and 9 as the highest. In addition, clients should consider priorities 0-4 as gradations of normal priority and priorities 5-9 as gradations of expedited priority. The JMS API does not require that a provider strictly implement priority ordering of messages; however, it should do its best to deliver expedited messages ahead of normal messages.

Declaration

```
mqint getJMSPriority() FMQCONST
                            throw (CJMSException *);
```

Returns

The default message priority

Throws

CJMSException

If the JMS provider fails to get the message priority due to some internal error.

getJMSRedelivered

Gets an indication of whether this message is being redelivered. If a client receives a message with the JMSRedelivered field set. It is likely, but not guaranteed, that this message was delivered earlier but that its receipt was not acknowledged at that time.

Declaration

```
mqboolean getJMSRedelivered() FMQCONST
                            throw (CJMSException *);
```

Returns

TRUE if this message is being redelivered

Throws

CJMSException

If the JMS provider fails to get the redelivered state due to some internal error.

getJMSReplyTo

Gets the Destination object to which a reply to this message should besent.

Declaration

```
CDestination *getJMSReplyTo() FMQCONST
                            throw (CJMSException *);
```

Returns

Destination to which to send a response to this message

Throws

CJMSException

If the JMS provider fails to get the JMSReplyTo destination due to some internal error.

getJMSTimestamp

Gets the message timestamp. The JMSTimestamp header field contains the time when the message was handed off to a provider to be sent. It is not the time when the message was actually transmitted, because the actual transmission may occur later due to transactions or other client-side queueing of messages. When a message is sent, JMSTimestamp is ignored.

Declaration

```
mqlong getJMSTimestamp() FMQCONST
                            throw (CJMSException *);
```

Returns

The timestamp of the message

Throws

CJMSException

If the JMS provider fails to get the timestamp due to someinternal error.

getJMSType

Gets the message type identifier supplied by the client when the message was sent.

Declaration

```
mqcstring getJMSType() FMQCONST
                            throw (CJMSException *);
```

Returns

The message type

Throws

CJMSException

If the JMS provider fails to get the message type due to some internal error.

getLongProperty

Returns the value of the long property with the specified name.

Declaration

```
mqlong getLongProperty(mqcstring propName) FMQCONST
                              throw (CJMSException *);
```

Parameters

PropName

The name of the long property

Returns

The long property value for the specified name

Throws

CJMSException

If the JMS provider fails to get the property value due to some internal error.

getObjectProperty

Returns the value of the object property with the specified name. This method can be used to return, in objectified format, an object that has been stored as a property in the message with the equivalent setObjectProperty method call, or its equivalent primitive settypeProperty method.

Declaration

```
mqobject getObjectProperty(mqcstring propName) FMQCONST
                          throw (CJMSException *);
```

Parameters

PropName

The name of the object property

Returns

The object property value with the specified name.

Throws

CJMSException

If the JMS provider fails to get the property value due to some internal error.

getPropertyNames

Returns an enumeration of the property names from the message object

Declaration

```
CEnumeration *getPropertyNames() FMQCONST
throw (CJMSException *);
```

Returns

An enumeration of the property names.

Throws

CJMSException

If the JMS provider fails to get the property value due to some internal error.

getShortProperty

Returns the value of the short property with the specified name.

Declaration

```
mqshort getShortProperty(mqcstring propName) FMQCONST
                            throw (CJMSException *);
```

Parameters

PropName

The name of the short property

Returns

The short property value for the specified name

Throws

CJMSException

If the JMS provider fails to get the property value due to some internal error.

getStringProperty

Returns the value of the String property with the specified name.

Declaration

```
mqcstring getStringProperty(mqcstring propName) FMQCONST
                            throw (CJMSException *);
```

Parameters

PropName

The name of the String property

Returns

The String property value for the specified name. If there is no property by this name, a null value is returned

Throws

CJMSException

If the JMS provider fails to get the property value due to some internal error.

getStringProperty_unicode

Returns the unicode string property with the specified name.

Declaration

```
mqcstring_unicode getStringProperty_unicode(mqcstring_unicode propName) FMQCONST
throw (CJMSException *);
```

Returns

The unicode string property value for the specified name. If there is no property by this name, a null value is returned

Throws

CJMSException

If the JMS provider fails to get the property value due to some internal error.

getMessageType

Returns the message type property value as mqint.

Declaration

```
mqint getMessageType()
throw (CJMSException *);
```

Returns

The mqint property value for the message type

Throws

CJMSException

If the JMS provider fails to get the property value due to some internal error.

propertyExists

Indicates whether a property value exists.

Declaration

```
mqboolean propertyExists(mqcstring propName) FMQCONST
                              throw (CJMSException *);
```

Parameters

PropName

The name of the property to test

Returns

TRUE if the property exists

Throws

CJMSException

If the JMS provider fails to determine if the property exists due to some internal error.

setBooleanProperty

Sets a boolean property value with the specified name into the message.

Declaration

```
void setBooleanProperty(mqcstring propName, mqboolean value)
                            throw (CJMSException *);
```

Parameters

name

The name of the boolean property value - the boolean property value to set

Throws

CJMSException

If the JMS provider fails to set the property due to some internal error.

setByteProperty

Sets a byte property value with the specified name into the message.

Declaration

```
void setByteProperty(mqcstring propName, mqbyte value)
                            throw (CJMSException *);
```

Parameters

name

The name of the byte property value - the byte property value to set

Throws

CJMSException

If the JMS provider fails to set the property due to some internal error.

setDoubleProperty

Sets a double property value with the specified name into the message.

Declaration

```
void setDoubleProperty(mqcstring propName, mqdouble value)
                            throw (CJMSException *);
```

Parameters

name

The name of the double property
value

The double property value to set

Throws

CJMSException

If the JMS provider fails to set the property due to some internal error.

setFloatProperty

Sets a float property value with the specified name into the message.

Declaration

```
void setFloatProperty(mqcstring propName, mqfloat value)
                            throw (CJMSException *);
```

Parameters

name

The name of the float property
value

The float property value to set

Throws

CJMSException

if the JMS provider fails to set the property due to some internal error.

setIntProperty

Sets an int property value with the specified name into the message.

Declaration

```
void setIntProperty(mqcstring propName, mqint value)
                            throw (CJMSException *);
```

Parameters

name

The name of the int property
value

The int property value to set

Throws

CJMSException

If the JMS provider fails to set the property due to some internal error.

setJMSCorrelationID

Sets the correlation ID for the message. A client can use the JMSCorrelationID header field to link one message with another. A typical use is to link a response message with its request message.

Declaration

```
void setJMSCorrelationID(mqcstring corrID)
                            throw (CJMSException *);
```

Parameters
correlationID

The message ID of a message being referred to

Throws

CJMSException

If the JMS provider fails to set the correlation ID due tosome internal error.

setJMSDeliveryMode

Sets the DeliveryMode value for this message. JMS providers set this field when a message is sent. This method can be used to change the value for a message that has been received.

Declaration

```
void setJMSDeliveryMode(mqint deliveryMode)
                            throw (CJMSException *);
```

Parameters
deliveryMode

The delivery mode for this message

Throws

CJMSException

If the JMS provider fails to set the delivery mode due to some internal error.

setJMSDestination

Sets the Destination object for this message. JMS providers set this field when a message is sent. This method can be used to change the value for a message that has been received.

Declaration

```
void setJMSDestination(CDestination *dest)
                            throw (CJMSException *);
```

Parameters

destination

The destination for this message

Throws

CJMSException

If the JMS provider fails to set the destination due to some internal error.

setJMSExpiration

Sets the message's expiration value. JMS providers set this field when a message is sent. This method can be used to change the value for a message that has been received.

Declaration

```
void setJMSExpiration(mqlong expiration)
                            throw (CJMSException *);
```

Parameters

expiration

The message's expiration time

Throws

CJMSException

If the JMS provider fails to set the message expiration dueto some internal error.

setJMSMessageID

Sets the message ID. JMS providers set this field when a message is sent. This method can be used to change the value for a message that has been received.

Declaration

```
void setJMSMessageID(mqcstring msgID)
                            throw (CJMSException *);
```

Parameters

msgID

The ID of the message

Throws

CJMSException

If the JMS provider fails to set the message ID due to some internal error.

setJMSPriority

Sets the priority level for this message. JMS providers set this field whena message is sent. This method can be used to change the value for a message that has been received.

Declaration

```
void setJMSPriority(mqint priority)
                            throw (CJMSException *);
```

Parameters

priority

The priority of this message

Throws

CJMSException

If the JMS provider fails to set the message priority due to some internal error.

setJMSRedelivered

Specifies whether this message is being redelivered. This field is set at the time the message is delivered. This method can be used to change the value for a message that has been received.

Declaration

```
void setJMSRedelivered(mqboolean redelivered)
                            throw (CJMSException *);
```

Parameters

redelivered

An indication of whether this message is being redelivered

Throws

CJMSException

If the JMS provider fails to set the redelivered state due to some internal error.

setJMSReplyTo

Sets the Destination object to which a reply to this message should be sent. The JMSReplyTo header field contains the destination where a reply to the current message should be sent. If it is null, no reply is expected. The destination may be either a Queue object or a Topic object. Messages sent with a null JMSReplyTo value may be a notification of some event, or they may just be some data the sender thinks is of interest. Messages with a JMSReplyTo value typically expect a response. A response is optional; it is up to the client to decide. These messages are called requests. A message sent in response to a request is called a reply. In some cases a client may wish to match a request it sent earlier with a reply it has just received. The client can use the JMSCorrelationID header field for this purpose.

Declaration

```
void setJMSReplyTo(CDestination *dest)
                            throw (CJMSException *);
```

Parameters

dest

Destination to which to send a response to this message

Throws

CJMSException

If the JMS provider fails to set the JMSReplyTo destinationdue to some internal error.

setJMSTimestamp

Sets the message timestamp. JMS providers set this field when a message is sent. This method can be used to change the value for a message that has been received.

Declaration

```
void setJMSTimestamp(mqlong timestamp)
                            throw (CJMSException *);
```

Parameters

timestamp

The timestamp for this message

Throws

CJMSException

If the JMS provider fails to set the timestamp due to some internal error.

setJMSType

Sets the message type. Some JMS providers use a message repository that contains the definitions of messages sent by applications. The JMSType header field may reference a message's definition in the provider's repository. The JMS API does not define a standard message definition repository, nor does it define a naming policy for the definitions it contains.

Declaration

```
void setJMSType(mqcstring type)
                            throw (CJMSException *);
```

Parameters

type

The message type

Throws

CJMSException

If the JMS provider fails to set the message type due to some internal error.

setLongProperty

Sets a long property value with the specified name into the message.

Declaration

```
void setLongProperty(mqcstring propName, mqlong value)
                            throw (CJMSException *);
```

Parameters

name

The name of the long property
value

The long property value to set.

Throws

CJMSException

If the JMS provider fails to set the property due to some internal error.

setObjectProperty

Sets a object property value with the specified name into the message.

This method works only for the objectified primitive object types (Integer, Double, Long...) and String objects.

Declaration

```
void setObjectProperty(mqcstring propName, mqobject value, mqint size)
                            throw (CJMSException *);
```

Parameters

name

The name of the object property
value

The object property value to set.

Throws

CJMSException

If the JMS provider fails to set the property due to some internal error.

setshortProperty

Sets a short property value with the specified name into the message.

Declaration

```
void setShortProperty(mqcstring propName, mqshort value)
throw (CJMSException *);
```

Parameters

PropName

The name of the short property
value

The short property value to set.

Throws

CJMSException

f the JMS provider fails to set the property due to some internal error.

setStringProperty

Sets a String property value with the specified name into the message.

Declaration

```
void setStringProperty(mqcstring propName, mqcstring value)
throw (CJMSException *);
```

Parameters

PropName

The name of the String property
value

The String property value to set.

Throws

CJMSException

If the JMS provider fails to set the property due to some internal error.

CMessageConsumer

A client uses a CMessageConsumer object to receive messages from a destination. A CMessageConsumer object is created by passing a CDestination object to a message-consumer creation method supplied by a session.

Base for

CTopicSubscriber CQueueReceiver

CMessageListener

A CMessageListener object is used to receive asynchronously delivered messages.

Passes a message to the listener.

Declaration

```
virtual void onMessage(CMessage *msg) = 0;
```

Parameters

message

The message passed to the listener

CServerSession

A CServerSession object is an application server object that is used by a server to associate a thread with a JMS session.

CServerSessionPool

A CServerSessionPool object is a single-threaded context for producing and consuming messages. It is considered a lightweight JMS object.

CSession

A CSession object is a single-threaded context for producing and consuming messages. It is considered a lightweight JMS object.

Base for

CTopicSession CQueueSession

With this acknowledgment mode, the session automatically acknowledges a client's receipt of a message either when the session has successfully returned from a call to receive or when the message listener the session has called to process the message successfully returns.

```
static const int CAUTO_ACKNOWLEDGE SET_CAUTO_ACKNOWLEDGE;
```

With this acknowledgment mode, the client acknowledges a consumed message by calling the message's acknowledge method.

```
static const int CCLIENT_ACKNOWLEDGE SET_CCLIENT_ACKNOWLEDGE;
```

This acknowledgment mode instructs the session to lazily acknowledge the delivery of messages

```
static const int CDUPS_OK_ACKNOWLEDGE SET_CDUPS_OK_ACKNOWLEDGE;
```

This value is returned from the method getAcknowledgeMode if the session is transacted.

```
static const int CSESSION_TRANSACTED SET_CSESSION_TRANSACTED;
```

CStreamMessage

A CStreamMessage object is used to send a stream of primitive types in C++ programming language. It is filled and read sequentially. It inherits from the CMessage interface and adds a stream message body.

Derives

CMessage

readBoolean

Reads a boolean from the stream message.

Declaration

```
mqboolean readBoolean() FMQCONST
                            throw (CJMSException *);
```

Returns

The Boolean value read

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

readByte

Reads a byte value from the stream message.

Declaration

```
mqbyte readByte() FMQCONST
                              throw (CJMSException *);
```

Returns

The next byte from the stream message as a 8-bit byte

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

readBytes

Reads a byte array field from the stream message into the specified value.

Declaration

```
mqint readBytes(mqbyteArray value, mqint length) FMQCONST
                              throw (CJMSException *);
```

Parameters
value

The buffer into which the data is read

Returns

The total number of bytes read into the buffer, or -1 if there is no more data because the end of the byte field has been reached

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

readChar

Reads a Unicode character value from the stream message.

Declaration

```
mqchar readChar() FMQCONST
                              throw (CJMSException *);
```

Returns

A Unicode character from the stream message

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

readDouble

Reads a double from the stream message.

Declaration

```
mqdouble readDouble() FMQCONST
                              throw (CJMSException *);
```

Returns

A double value from the stream message

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

readFloat

Reads a float from the stream message.

Declaration

```
mqfloat readFloat() FMQCONST
                              throw (CJMSException *);
```

Returns

A float value from the stream message.

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

readInt

Reads a 32-bit integer from the stream message.

Declaration

```
mqint readInt() FMQCONST
throw (CJMSException *);
```

Returns

A 32-bit integer value from the stream message, interpreted as an integer

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

readLong

Reads a 64-bit integer from the stream message.

Declaration

```
mqlong readLong() FMQCONST
                              throw (CJMSException *);
```

Returns

A 64-bit integer value from the stream message, interpreted as a long.

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

readShort

Reads a 16-bit integer from the stream message.

Declaration

```
mqshort readShort() FMQCONST
                              throw (CJMSException *);
```

Returns

A 16-bit integer from the stream message.

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

readString

Reads a String from the stream message.

Declaration

```
mqcstring readString() FMQCONST
                              throw (CJMSException *);
```

Returns

A mqcstring from the stream message

Note: This string should be manually freed by the user, as it is not freed by deleting the Message Object.

Throws

CJMSException

If the JMS provider fails to read the message due to some internal error.

reset

Puts the message body in read-only mode and repositions the stream to the beginning.

Declaration

```
void reset()
                            throw (CJMSException *);
```

Throws

CJMSException

If the JMS provider fails to reset the message due to some internal error.

writeBoolean

Writes a Boolean to the stream message.

Declaration

```
void writeBoolean(mqboolean value)
                            throw (CJMSException *);
```

Parameters

value

The Boolean value to be written.

Throws

CJMSException

If the JMS provider fails to write the message due to some internal error.

writeByte

Writes a byte to the stream message.

Parameters

value

The byte value to be written.

Declaration

```
void writeByte(mqbyte value)
                            throw (CJMSException *);
```

Throws

CJMSException

If the JMS provider fails to write the message due to some internal error.

writeBytes

Writes a byte array field to the stream message. The byte array value is written to the message as a byte array field. Consecutively written byte array fields are treated as two distinct fields when the fields are read.

Declaration

```
void writeBytes(mqbyteArray value, mqint length)
                            throw (CJMSException *);
```

Parameters

value

The byte array value to be written
length

length of the bytes to be written

Throws

CJMSException

If the JMS provider fails to write the message due to some internal error.

writeBytes

Writes a portion of a byte array as a byte array field to the stream message. The a portion of the byte array value is written to the message as a byte array field. Consecutively written byte array fields are treated as two distinct fields when the fields are read.

Declaration

```
void writeBytes(mqbyteArray value, mqint offset, mqint length)
                            throw (CJMSException *);
```

Parameters

value

The byte array value to be written
offset

The initial offset within the byte array
length

The number of bytes to be written

Throws

CJMSException

If the JMS provider fails to write the message due to some internal error.

writeChar

Writes a char to the stream message.

Declaration

```
void writeChar(mqchar value)
                            throw (CJMSException *);
```

Parameters

value

The char value to be written

Throws

CJMSException

If the JMS provider fails to write the message due to some internal error.

writeDouble

Writes a double to the stream message.

Declaration

```
void writeDouble(mqdouble value)
                            throw (CJMSException *);
```

Parameters

value

The double value to be written

Throws

CJMSException

if the JMS provider fails to write the message due to some internal error.

writeFloat

Writes a float to the stream message.

Declaration

```
void writeFloat(mqfloat value)
                            throw (CJMSException *);
```

Parameters

value

The float value to be written

Throws

CJMSException

If the JMS provider fails to write the message due to some internal error.

writeInt

Writes an int to the stream message.

Declaration

```
void writeInt(mqint value)
                            throw (CJMSException *);
```

Parameters

value

The int value to be written

Throws

CJMSException

If the JMS provider fails to write the message due to some internal error.

writeLong

Writes a long to the stream message.

Declaration

```
void writeLong(mqlong value)
                            throw (CJMSException *);
```

Parameters

value

The long value to be written

Throws

CJMSException

If the JMS provider fails to write the message due to some internal error.

writeShort

Writes a short to the stream message.

Declaration

```
void writeShort(mqshort value)
                            throw (CJMSException *);
```

Parameters

value

The short value to be written

Throws

CJMSException

If the JMS provider fails to write the message due to some internal error.

writeString

Writes a String to the stream message.

Declaration

```
void writeString(mqcstring value)
                            throw (CJMSException *);
```

Parameters

value

The String value to be written

Throws

CJMSException

If the JMS provider fails to write the message due to some internal error.

CTextMessage

A CTextMessage object is used to send a message containing a mqcstring. It inherits from the CMessage interface and adds a text message body.

Derives

CMessage

getText

Gets the mqcstring containing this message's data. The default value is null.

Declaration

```
mqcstring getText()
                              throw (CJMSException *);
```

Returns

The String containing the message's data

Throws

CJMSException

if the JMS provider fails to get the text due to some internal error.

setText

Sets the mqcstring containing this message's data.

Declaration

```
void setText(mqcstring)
                              throw (CJMSException *;
```

Parameters

mqcstring

The String containing the message's data

Throws

CJMSException

If the JMS provider fails to set the text due to some internal error.

Naming and Lookup (JNDI)

CInitialContext

This class is the starting context for performing naming operations.

Constructor

Constructs an initial context using the supplied environment.

Declaration

```
CInitialContext(CHashTable  *env)
                            throw (CJMSException*);
```

Parameters

env

Hashtable Contains the environment information.

Lookup

Retrieves the named object.

Declaration

```
fmqobject Lookup(mqcstring adminObjectName)
                            throw (CJMSException *);
```

Parameters

adminObjectName

Name of the Admin Object.

Returns

The named object as fmqobject which can be cast to the required admin object.

LookupQCF

Retrieves the Queue Connection Factory object in serverless mode. If server is present, it looks up as normal Lookup function.

Declaration

```
fmqobject LookupQCF(mqcstring adminObjectName)
                            throw (CJMSException *);
```

Parameters

adminObjectName

Name of the Admin Object.

Returns

The named object as fmqobject which can be cast to the Queue Connection Factory object.

LookupTCF

Retrieves the Topic Connection Factory object in serverless mode. If server is present, it looks up as normal Lookup function.

Declaration

```
fmqobject LookupTCF(mqcstring adminObjectName)
                            throw (CJMSException *);
```

Parameters

adminObjectName

Name of the Admin Object.

Returns

The named object as fmqobject which can be cast to the Topic Connection Factory object.

PTP

CQueue

A CQueue object encapsulates a provider-specific queue name. It is the way a client specifies the identity of a queue to JMS API methods. For those methods that use a CDestination as a parameter, a CQueue object is used as an argument.

Derives

CDestination

getQueueName

Gets the name of this queue. Clients that depend upon the name are not portable.

Declaration

```
mqcstring getQueueName()
                            throw (CJMSException *);
```

Returns

The queue name

Throws

CJMSException

If the JMS provider implementation of Queue fails to return the queue name due to some internal error.

toString

Returns a mqcstring representation of this object.

Declaration

```
mqcstring toString()
                              throw (CJMSException *);
```

Returns

The provider-specific identity values for this queue.

CQueueConnection

A CQueueConnection object is an active connection to a point-to-point JMS provider. A client uses a CQueueConnection object to create one or more CQueueSession objects for producing and consuming messages.

Derives

CConnection

createQueueSession

Creates a QueueSession object.

Declaration

```
CQueueSession *createQueueSession(mqboolean transacted,
mqint acknowledgeMode)
                              throw (CJMSException *);
```

Parameters

transacted

Indicates whether the session is transacted acknowledge.

acknowledgeMode

Indicates whether the consumer or the client will acknowledge any messages it receives; ignored if the session is transacted. Legal values are CJMSSession.CAUTO_ACKNOWLEDGE, CJMSSession.CCLIENT_ACKNOWLEDGE, and CJMSSession.CDUPS_OK_ACKNOWLEDGE.

Returns

A newly created queue session

Throws

CJMSException

If the QueueConnection object fails to create a session due to some internal error or lack of support for the specific transaction and acknowledgement mode.

close

Closes the connection. Since a provider typically allocates significant resources outside the JVM on behalf of a connection, clients should close these resources when they are not needed. Relying on garbage collection to eventually reclaim these resources may not be timely enough. There is no need to close the sessions, producers, and consumers of a closed connection. Closing a connection causes all temporary destinations to be deleted.

Declaration

```
void close()

throw (CJMSException *);
```

Throws

CJMSException

If the JMS provider fails to close the connection due to some internal error. For example, a failure to release resources or to close a socket connection can cause this exception to be thrown.

getClientID

Gets the client identifier for this connection. This value is specific to the JMS provider. It is either preconfigured by an administrator in a ConnectionFactory object or assigned dynamically by the application by calling the setClientID method.

Declaration

```
mqcstring getClientID() FMQCONST
                              throw (CJMSException *);
```

Returns

The unique client identifier

Throws

CJMSException

If the JMS provider fails to return the client ID for this connection due to some internal error.

setAdvisoryMessageListener

Sets an advisory listener for this connection. If a JMS provider detects a seriousproblem with a connection with server as in connection breakup and activation of the reconnect thread, it informs the connection's AdvisoryMessageListener, if has been registered. It does this by calling the listener's onAdvisoryMessage method, passing it a CAdvisoryMessage object describing the problem. An advisory listener allows a client to be notified of a server connection failures asynchronously.

Declaration

```
void setAdvisoryMessageListener(CAdvisoryMsgListener *advMsgListener)
                          throw (CJMSException *);
```

Parameters

advMsgListener

The advisory message listener object.

Throws

CJMSException

If the JMS provider fails to set the exception listener for this connection.

setClientID

Sets the client identifier for this connection. The preferred way to assign a JMS client's client identifier is for it to be configured in a client-specific CConnectionFactory object and transparently assigned to the Connection object it creates.

Declaration

```
void setClientID(mqcstring clientID)
                            throw (CJMSException *);
```

Parameters

clientID

The unique client identifier

Throws

CJMSException

If the JMS provider fails to set the client ID for this connection due to some internal error.

start

Starts (or restarts) a connection's delivery of incoming messages. A call to start on a connection that has already been started is ignored.

Declaration

```
void start()
throw (CJMSException *);
```

Throws

CJMSException

If the JMS provider fails to start message delivery due to some internal error.

stop

Temporarily stops a connection's delivery of incoming messages. Delivery can be restarted using the connection's start method. When the connection is stopped, delivery to all the connection's message consumers is inhibited: synchronous receives block, and messages are not delivered to message listeners. This call blocks until receives and/or message listeners in progress have completed.

Declaration

```
void stop()
                              throw (CJMSException *);
```

Throws

CJMSException

If the JMS provider fails to stop message delivery due to some internal error.

getExceptionListener

Gets the ExceptionListener object for this connection. Not every Connection has an ExceptionListener associated with it.

Declaration

```
CExceptionListener *getExceptionListener() FMQCONST
                              throw (CJMSException *);
```

Returns

The ExceptionListener for this connection, or null. if no ExceptionListener is associated with this connection.

Throws

CJMSException

If the JMS provider fails to get the ExceptionListener for this connection.

setExceptionListener

Sets an exception listener for this connection. If a JMS provider detects a serious problem with a connection, it informs the connection's ExceptionListener, if one has been registered. It does this by calling the listener's onException method, passing it a CJMSException object describing the problem. An exception listener allows a client to be notified of a problem asynchronously. Some connections only consume messages, so they would have no other way to learn their connection has failed. A connection serializes execution of its ExceptionListener.

Declaration

```
void setExceptionListener(CExceptionListener *exceptionListener)
                              throw (CJMSException *);
```

Parameters

listener

The exception listener

Throws

CJMSException

If the JMS provider fails to set the exception listener for this connection.

CQueueConnectionFactory

A client uses a CQueueConnectionFactory object to create CQueueConnection objects with a point-to-point JMS provider.

Derives

CConnectionFactory

createQueueConnection

Creates a queue connection with the default user identity. The connection is created in stopped mode. No messages will be delivered until the Connection.start method is explicitly called.

Declaration

```
CQueueConnection *createQueueConnection()
                            throw (CJMSException *);
```

Returns

A newly created queue connection

Throws

CJMSException

If the JMS provider fails to create the queue connection due to some internal error.

createQueueConnection

Creates a queue connection with the specified user identity. The connection is created in stopped mode. No messages will be delivered until the Connection.start method is explicitly called.

Declaration

```
CQueueConnection *createQueueConnection(mqcstring username,

mqcstring password)
                            throw (CJMSException *);
```

Parameters

userName

The caller's user name
password

The caller's password

Returns

A newly created queue connection

Throws

CJMSException

If the JMS provider fails to create the queue connection due to some internal error.

CQueueReceiver

A client uses a CQueueReceiver object to receive messages that have been delivered to a queue.

Derives

CMessageConsumer

getQueue

Gets the Queue associated with this queue receiver.

Declaration

```
CQueue *getQueue()
                              throw (CJMSException *);
```

Returns

This receiver's Queue

Throws

CJMSException

If the JMS provider fails to get the queue for this queue receiver due to some internal error

close

Closes the message consumer. Since a provider may allocate some resources on behalf of a MessageConsumer outside the Java virtual machine, clients should close them when they are not needed. Relying on garbage collection to eventually reclaim these resources may not be timely enough. This call blocks until a receive or message listener in progress has completed. A blocked message consumer receive call returns null when this message consumer is closed.

Declaration

```
void close()
                              throw (CJMSException *);
```

Throws

CJMSException

If the JMS provider fails to close the consumer due to some internal error.

getMessageListener

Gets the message consumer's MessageListener.

Declaration

```
CMessageListener *getMessageListener() FMQCONST
                              throw (CJMSException *);
```

Returns

The listener for the message consumer, or null if no listener is set

Throws

CJMSException

If the JMS provider fails to get the message listener due to some internal error.

getMessageSelector

Gets this message consumer's message selector expression.

Declaration

```
mqcstring getMessageSelector() FMQCONST
                              throw (CJMSException *);
```

Returns

This message consumer's message selector, or null if no message selector exists for the message consumer (that is, if the message selector was not set or was set to null or the empty mqcstring)

Throws

CJMSException

If the JMS provider fails to get the message selector due to some internal error.

receive

Receives the next message produced for this message consumer. This call blocks indefinitely until a message is produced or until this message consumer is closed. This is done within a transaction, the consumer retains the message until the transaction commits.

Declaration

```
CMessage *receive()
                              throw (CJMSException *);
```

Returns

The next message produced for this message consumer, or null if this message consumer is concurrently closed.

Throws

CJMSException

If the JMS provider fails to receive the next message due to some internal error.

recieve

Receives the next message that arrives within the specified timeout interval. This call blocks until a message arrives, the timeout expires, or this message consumer is closed. A timeout of zero never expires, and the call blocks indefinitely.

Declaration

```
CMessage *receive(mqlong timeout)
                            throw (CJMSException *);
```

Parameters

timeout]

The timeout value (in milliseconds)

Returns

The next message produced for this message consumer, or null if the timeout expires or this message consumer is concurrently closed.

Throws

CJMSException

If the JMS provider fails to receive the next message due to some internal error.

receiveNoWait

Receives the next message if one is immediately available.

Declaration

```
CMessage *receiveNoWait()
                            throw (CJMSException *);
```

Returns

The next message produced for this message consumer, or null if one is not available

Throws

CJMSException

If the JMS provider fails to receive the next message due to some internal error.

setMessageListener

Sets the message consumer's MessageListener. Setting the message listener to null is equivalent to unsetting the message listener for the message consumer. The effect of calling MessageConsumer. setMessageListener while messages are being consumed by an existing listener or the consumer is being used to consume messages synchronously is undefined.

Declaration

```
void setMessageListener(CMessageListener *messageListener)
                            throw (CJMSException *);cppQueue
```

Parameters

`listener`

The listener to which the messages are to be delivered

Throws

`CJMSException`

If the JMS provider fails to set the message listener due to some internal error.

CQueueRequestor

The CQueueRequestor helper class simplifies making service requests.

close

Closes the QueueRequestor and its session. Since a provider may allocate some resources on behalf of a QueueRequestor outside the Java virtual machine, clients should close them when they are not needed. Relying on garbage collection to eventually reclaim these resources may not be timely enough.

Declaration

```
void close()
                            throw (CJMSException *);
```

Throws

`CJMSException`

If the JMS provider fails to close the QueueRequestor due to some internal error.

request

Sends a request and waits for a reply. The temporary queue is used for the JMSReplyTo destination, and only one reply per request is expected.

Declaration

```
CMessage *request(CMessage *msg)
                            throw (CJMSException *);
```

Parameters

message

The message to send

Returns

The reply message

Throws

CJMSException

If the JMS provider fails to complete the request due to some internal error.

request

Sends a request and waits for a reply until timeout. The temporary queue is used for the JMSReplyTo destination, and only one reply per request is expected.

Declaration

```
CMessage *request(CMessage *msg, mqlong timeout)
                            throw (CJMSException *);
```

Parameters

message

The message to send
timeout

Time until which to wait for message

Returns

The reply message

Throws

CJMSException

If the JMS provider fails to complete the request due to some internal error.

CQueueBrowser

A client uses a CQueueBrowser object to look at messages on a queue without removing them.

close

Closes the QueueBrowser. Since a provider may allocate some resources on behalf of a QueueBrowser outside the Java virtual machine, clients should close them when they are not needed. Relying on garbage collection to eventually reclaim these resources may not be timely enough.

Declaration

```
void close()
                              throw (CJMSException *);
```

Throws

CJMSException

If the JMS provider fails to close this browser due to some internal error.

getMessageSelector

Gets this queue browser's message selector expression.

Declaration

```
mqcstring getMessageSelector()
                              throw (CJMSException *);
```

Returns

This queue browser's message selector, or null if no message selector exists for the message consumer (that is, if the message selector was not set or was set to null or the empty mqcstring)

Throws

CJMSException

If the JMS provider fails to get the message selector for this browser due to some internal error.

getQueue

Gets the queue associated with this queue browser.

Declaration

```
CQueue *getQueue()
                              throw (CJMSException *);
```

Returns

The queue

Throws

CJMSException

If the JMS provider fails to get the queue associated with this browser due to some internal error.

CQueueSender

A client uses a CQueueSender object to send messages to a queue.

Derives

CMessageProducer

getQueue

Gets the queue associated with this QueueSender.

Declaration

```
CQueue *getQueue()
                              throw (CJMSException *);
```

Returns

This sender's queue

Throws

CJMSException

If the JMS provider fails to get the queue for this QueueSender due to some internal error.

close

Closes the message producer. Since a provider may allocate some resources on behalf of a MessageProducer outside the Java virtual machine, clients should close them when they are not needed. Relying on garbage collection to eventually reclaim these resources may not be timely enough.

Declaration

```
void close()
                              throw (CJMSException *);
```

Throws

CJMSException

If the JMS provider fails to close the producer due to some internal error.

getDeliveryMode

Gets the producer's default delivery mode.

Declaration

```
mqint getDeliveryMode() FMQCONST
                              throw (CJMSException *);
```

Returns

The message delivery mode for this message producer

Throws

CJMSException

If the JMS provider fails to get the delivery mode due to some internal error.

getDestination

Gets the destination associated with this MessageProducer.

Declaration

```
CDestination *getDestination() FMQCONST
                              throw (CJMSException *);
```

Returns

This producer's Destination

Throws

CJMSException

If the JMS provider fails to get the destination for this MessageProducer due to some internal error.

getDisableMessageID

Gets an indication of whether message IDs are disabled.

Declaration

```
mqboolean getDisableMessageID() FMQCONST
                              throw (CJMSException *);
```

Returns

An indication of whether message IDs are disabled

Throws

CJMSException

if the JMS provider fails to determine if message IDs are disabled due to some internal error.

getDisableMessageTimestamp

Gets an indication of whether message timestamps are disabled

Declaration

```
mqboolean getDisableMessageTimestamp() FMQCONST
```

```
                                throw (CJMSException *);
```

Returns

An indication of whether message timestamps are disabled

Throws:

CJMSException

If the JMS provider fails to determine if timestamps are

disabled due to some internal error.

getPriority

Gets the producer's default priority.

Declaration

```
mqint getPriority() FMQCONST
                                throw (CJMSException *);
```

Returns

The message priority for this message producer.

Throws

CJMSException

If the JMS provider fails to get the priority due to some internal error.

getTimeToLive

Gets the default length of time in milliseconds from its dispatch time that a produced message should be retained by the message system.

Declaration

```
mqlong getTimeToLive() FMQCONST
                                throw (CJMSException *);
```

Returns

The message time to live in milliseconds; zero is unlimited

Throws

CJMSException

If the JMS provider fails to get the time to live due to some internal error.

send

Sends a message to a destination for an unidentified message producer. Uses the MessageProducer's default delivery mode, priority, and time to live.

Declaration

```
void send(CDestination *dest, CMessage *msg)
                            throw (CJMSException *);
```

Parameters

destination

The destination to send this message to
message

The message to send.

Throws

CJMSException

If the JMS provider fails to send the message due to some internal error.

send

Sends a message to a destination for an unidentified message producer, specifying delivery mode, priority and time to live. Typically, a message producer is assigned a destination at creation time; however, the JMS API also supports unidentified message producers, which require that the destination be supplied every time a message is sent.

Declaration

```
void send(CDestination *dest, CMessage *msg, mqint deliveryMode,
mqint priority, mqint timeToLive) throw (CJMSException *);
```

Parameters

destination

The destination to send this message to
message

The message to send
deliveryMode

The delivery mode to use
priority

The priority for this message
timeToLive

The message's lifetime (in milliseconds)

Throws

CJMSException

If the JMS provider fails to send the message due to some internal error.

send

Sends a message to the queue. Uses the QueueSender's default delivery mode, priority, and time to live.

Declaration

```
void send(CMessage *msg)
                            throw (CJMSException *);
```

Parameters

message

The message to send

Throws

CJMSException

If the JMS provider fails to send the message due to some internal error.

send

Sends a message to the queue, specifying delivery mode, priority, and time to live.

Declaration

```
void send(CMessage *msg, mqint deliveryMode, mqint priority,
mqlong timeToLive) throw (CJMSException *);
```

Parameters

message

The message to send
deliveryMode

The delivery mode to use
priority

The priority for this message
timeToLive

The message's lifetime (in milliseconds)

Throws

CJMSException

If the JMS provider fails to send the message due to some internal error.

setDeliveryMode

Sets the producer's default delivery mode. Delivery mode is set to PERSISTENT by default.

Declaration

```
void setDeliveryMode(mqint deliveryMode)
                            throw (CJMSException *);
```

Parameters

deliveryMode

The message delivery mode for this message producer; legal values are DeliveryMode.CNON_PERSISTENT and DeliveryMode.CPERSISTENT

Throws

CJMSException

If the JMS provider fails to set the delivery mode due to some internal error.

setDisableMessageID

Sets whether message IDs are disabled. Message IDs are enabled by default.

Declaration

```
void setDisableMessageID(mqboolean value)
                            throw (CJMSException *);
```

Parameters

value

Indicates if message IDs are disabled

Throws

CJMSException

If the JMS provider fails to set message ID to disabled due

to some internal error.

setDisableMessageTimeStamp

Sets whether message timestamps are disabled. Message timestamps are enabled by default.

Declaration

```
void setDisableMessageTimestamp(mqboolean value)
                            throw (CJMSException *);
```

Parameters

value

Indicates if message timestamps are disabled

Throws

CJMSException

If the JMS provider fails to set timestamps to disabled due to some internal error.

setPriority

Sets the producer's default priority.The JMS API defines ten levels of priority value, with 0 as the lowest priority and 9 as the highest. Clients should consider priorities 0-4 as gradations of normal priority and priorities 5-9 as gradations of expedited priority. Priority is set to 4 by default.

```
void setPriority(mqint defaultPriority)
                            throw (CJMSException *);
```

Parameters

defaultPriority

The message priority for this message producer; must be a value between 0 and 9

Throws

CJMSException

If the JMS provider fails to set the priority due to some internal error.

setTimeToLive

Sets the default length of time in milliseconds from its dispatch time that a produced message should be retained by the message system. Time to live is set to zero by default.

Declaration

```
void setTimeToLive(mqlong timeToLive)
                            throw (CJMSException *);
```

Parameters

timeToLive

The message time to live in milliseconds; zero is unlimited

Throws

CJMSException

If the JMS provider fails to set the time to live due to some internal error.

CQueueSession

A CQueueSession object provides methods for creating CQueueReceiver, CQueueSender, CQueueBrowser, and CTemporaryQueue objects.

Derives

CJMSSession

close

Closes the session.

Declaration

```
void close()
throw (CJMSException *);
```

Throws

CJMSException

If the JMS provider fails to close the session due to some internal error.

commit

Commits all messages done in this transaction and releases any locks currently held.

Declaration

```
void commit()
                            throw (CJMSException *);
```

Throws

CJMSException

If the JMS provider fails to commit the transaction due to some internal error.

createBrowser

Creates a QueueBrowser object to peek at the messages on the specified queue.

Declaration

```
CQueueBrowser *createBrowser(CQueue *q)
                            throw (CJMSException *);
```

Specified by

createBrowser

in interface Session.

Parameters

queue

The Queue to access

Returns

A new queue browser

Throws

CJMSException

If the session fails to create a browser due to some internal error.

createBrowser

Creates a QueueBrowser object to peek at the messages on the specified queue using a message selector.

Declaration

```
CQueueBrowser *createBrowser(CQueue *q, mqcstring messageSelector)
                             throw (CJMSException *);
```

Specified by

CreateBrowser in interface Session

Parameters

queue

The Queue to access
messageSelector

Only messages with properties matching the message selector expression are delivered. A value of null or an empty mqcstring indicates that there is no message selector for the message consumer.

Throws

CJMSException

If the session fails to create a browser due to some internal error.

createReceiver

Creates a CQueueReceiver object to receive the messages on the specified queue.

Declaration

```
CQueueReceiver *createReceiver(CQueue *queue)
                             throw (CJMSException *);
```

Parameters

queue

The Queue to receive messages from

Returns

A new queue receiver

Throws

CJMSException

If the session fails to create a receiver due to some internal error.

createReceiver

Creates a CQueueReceiver object to receive messages on the specified queue using a message selector.

Declaration

```
CQueueReceiver *createReceiver(CQueue *queue, mqcstring messageSelector)
                               throw (CJMSException *);
```

Parameters

queue

The Queue to receive messages from

messageSelector

Only messages with properties matching the message selector expression are delivered. A value of null or an empty mqcstring indicates that there is no message selector for the message consumer.

Throws

CJMSException

If the session fails to create a receiver due to some internal error.

createSender

Creates a CQueueSender object to send messages to the specified queue.

Declaration

```
CQueueSender *createSender(CQueue *queue)
                               throw (CJMSException *);
```

Parameters

queue

The Queue to send messages to

Returns

A new queue sender

Throws

CJMSException

If the session fails to create a sender due to some internal error.

createBytesMesage

Creates a BytesMessage object. A BytesMessage object is used to send a message containing a stream of uninterpreted bytes.

Declaration

```
CBytesMessage *createBytesMesage()
throw (CJMSException *);
```

Returns

A BytesMessage object

Throws

CJMSException

If the JMS provider fails to create this message due to some internal error.

createMapMessage

Creates a MapMessage object. A MapMessage object is used to send a self-defining set of name-value pairs, where names are String objects and values are primitive values in the Java programming language.

Declaration

```
CMapMessage *createMapMessage()
throw (CJMSException *);
```

Returns

A MapMessage object

Throws

CJMSException

If the JMS provider fails to create this message due tosome internal error.

createQueue

Creates a queue identity given a Queue name.

Declaration

```
CQueue *createQueue(mqcstring queueName)
                            throw (CJMSException *);
```

Parameters

queueName

The name of this Queue

Returns

A Queue with the given name

Throws

CJMSException

If the session fails to create a queue due to some internal error.

createStreamMessage

Creates a StreamMessage object. A StreamMessage object is used to send a self-defining stream of primitive values in the Java programming language.

Declaration

```
CStreamMessage *createStreamMessage()
                            throw (CJMSException *);
```

Throws

CJMSException

If the JMS provider fails to create this message due tosome internal error.

createTemporaryQueue

Creates a TemporaryQueue object. Its lifetime will be that of the QueueConnection unless it is deleted earlier.

Declaration

```
CTemporaryQueue *createTemporaryQueue()
                            throw (CJMSException *);
```

Returns

A temporary queue identity

Throws

CJMSException

If the session fails to create a temporary queue due tosome internal error.

createTextMessage

Creates a TemporaryQueue object. Its lifetime will be that of the QueueConnection unless it is deleted earlier.

Declaration

```
CTextMessage *createTextMessage()
                            throw (CJMSException *);
```

Returns

A temporary queue identity

Throws

CJMSException

If the session fails to create a temporary queue due to some internal error.

createTextMessage

Creates an initialized TextMessage object. A TextMessage object is used to send a message containing a String.

Declaration

```
CTextMessage *createTextMessage(mqcstring text)
                            throw (CJMSException *);
```

Parameters

text

The mqcstring used to initialize this message

Returns

A TextMessage object

Throws

CJMSException

If the JMS provider fails to create this message due tosome internal error.

getMessageListener

Returns the session's distinguished message listener (optional).

Declaration

```
CMessageListener *getMessageListener()
                              throw (CJMSException *);
```

Returns

The message listener associated with this session

Throws

CJMSException

If the JMS provider fails to get the message listener due to an internal error.

recover

Stops message delivery in this session, and restarts message delivery with the oldest unacknowledged message.

Declaration

```
void recover()
                              throw (CJMSException *);
```

Throws

CJMSException

If the JMS provider fails to stop and restart message delivery due to some internal error.

rollback

Rolls back any messages done in this transaction and releases any lockscurrently held.

Declaration

```
void rollback()
                              throw (CJMSException *);
```

Throws

CJMSException

If the JMS provider fails to roll back the transaction due to some internal error.

run

Optional operation, intended to be used only by Application Servers, not byordinary JMS clients.

Declaration

```
void run()
                              throw (CJMSException *);
```

setMessageListener

Sets the session's distinguished message listener.

Declaration

```
void setMessageListener(CMessageListener *messageListener)
                            throw (CJMSException *);
```

Parameters

`listener`

The message listener to associate with this session

Throws

`CJMSException`

If the JMS provider fails to set the message listener due to an internal error.

CTemporaryQueue

A CTemporaryQueue object is a unique CQueue object created for the duration of a CConnection. It is a system-defined queue that can be consumed only by the CConnection that created it.

Derives

`CDestination`

remove

Deletes this temporary queue. If there are existing receivers still using it, a CJMSException will be thrown.

Declaration

```
void remove()
throw (CJMSException *);
```

Throws

`CJMSException`

If the JMS provider fails to delete the temporary queue due to some internal error.

Publish/Subscribe

CTemporaryTopic

A CTemporaryTopic object is a unique CTopic object created for the duration of a CConnection. It is a system-defined topic that can be consumed only by the CConnection that created it.

Derives

CDestination

remove

Deletes this temporary topic. If there are existing subscribers still using it, a CJMSException will be thrown.

Declaration

```
void remove()
                              throw (CJMSException *);
```

Throws

CJMSException

If the JMS provider fails to delete the temporary topic due to some internal error.

CTopic

A CTopic object encapsulates a provider-specific topic name. It is the way a client specifies the identity of a topic to JMS API methods. For those methods that use a CDestination as a parameter, a CTopic object may used as an argument.

Derives

CDestination

getTopicName

Gets the name of this topic. Clients that depend upon the name are not portable.

Declaration

```
mqcstring getTopicName()
                              throw (CJMSException *);
```

Returns

The topic name

Throws

CJMSException

If the JMS provider implementation of Topic fails to return the topic name due to some internal error.

toString

Returns a mqcstring representation of this object.

Declaration

```
mqcstring toString()
                              throw (CJMSException *);If
```

Returns

The provider-specific identity values for this topic.

Throws

CJMSException

If the TopicConnection object fails to return the provider-specific values due to some internal error.

CTopicConnection

CTopicConnection object is an active connection to a publish/subscribe JMS provider. A client uses a CTopicConnection object to create one or more CTopicSession objects for producing and consuming messages.

Derives

CConnection

createTopicSession

Creates a TopicSession object.

Declaration

```
CTopicSession *createTopicSession(mqboolean transacted,

mqint acknowledgeMode)
                              throw (CJMSException *);
```

Parameters

transacted

Indicates whether the session is transacted
acknowledgeMode

Indicates whether the consumer or the client will acknowledge any messages it receives; ignored if the session is transacted.

Returns

A newly created topic session

Throws

CJMSException

If the TopicConnection object fails to create a session due to some internal error or lack of support for the specific transaction and acknowledgement mode.

close

Closes the connection.

Declaration

```
void close()
                              throw (CJMSException *);
```

Throws

CJMSException - if the JMS provider fails to close the connection due to some internal error.

getClientID

Gets the client identifier for this connection.

Declaration

```
mqcstring getClientID() FMQCONST
                              throw (CJMSException *);
```

Returns

The unique client identifier

Throws

CJMSException

If the JMS provider fails to return the client ID for this connection due to some internal error.

setAdvisoryMessageListener

Sets an advisory listener for this connection. If a JMS provider detects a seriousproblem with a connection with server as in connection breakup and activation of the reconnect thread, it informs the connection's AdvisoryMessageListener, ifhas been registered. It does this by calling the listener's onAdvisoryMessage method, passing it a CAdvisoryMessage object describing the problem. An advisory listener allows a client to be notified of a server connection failures asynchronously.

Declaration

```
void setAdvisoryMessageListener(CAdvisoryMsgListener *advMsgListener)
                         throw (CJMSException *);
```

Parameters

advMsgListener

The advisory message listener object.

Throws

CJMSException

If the JMS provider fails to set the exception listener for this connection.

setClientID

Sets the client identifier for this connection.

Declaration /

```
void setClientID(mqcstring clientID)
                        throw (CJMSException *);
```

Parameters

clientID

The unique client identifier

Throws

CJMSException

If the JMS provider fails to set the client ID for this connection due to some internal error.

start

Starts (or restarts) a connection's delivery of incoming messages. A call to start on a connection that has already been started is ignored.

Declaration

```
void start()
                            throw (CJMSException *);
```

Throws

CJMSException -

If the JMS provider fails to start message delivery due to some internal error.

stop

Temporarily stops a connection's delivery of incoming messages. Delivery can be restarted using the connection's start method.

Declaration

```
void stop()
                            throw (CJMSException *);
```

Throws

CJMSException

If the JMS provider fails to stop message delivery due to some internal error.

getExceptionListener

Gets the ExceptionListener object for this connection. Not every Connectionhas an ExceptionListener associated with it.

Declaration

```
CExceptionListener *getExceptionListener() FMQCONST
                            throw (CJMSException *);
```

Returns

The ExceptionListener for this connection, or null. if no ExceptionListeneris associated with this connection.

Throws

CJMSException

If the JMS provider fails to get the ExceptionListener for this connection.

setExceptionListener

Sets an exception listener for this connection.

Declaration

```
void setExceptionListener(CExceptionListener *exceptionListener)
                            throw (CJMSException *);
```

Parameters

listener

The exception listener

Throws

CJMSException

If the JMS provider fails to set the exception listener for this connection.

CTopicConnectionFactory

A client uses a CTopicConnectionFactory object to create CTopicConnection objects with a publish/subscribe JMS provider.

Derives

CConnectionFactory

createTopicConnection

Creates a topic connection with the default user identity. The connection is created in stopped mode. No messages will be delivered until the Connection.start method is explicitly called.

Declaration

```
CTopicConnection *createTopicConnection()
                              throw (CJMSException *);
```

Returns

A newly created topic connection

Throws

CJMSException

If the JMS provider fails to create a topic connection due to some internal error.

createTopicConnection

Creates a topic connection with the specified user identity. The connectionis created in stopped mode. No messages will be delivered until the Connection.start method is explicitly called.

Declaration

```
CTopicConnection *createTopicConnection(mqcstring username, mqcstring password)
                              throw (CJMSException *);
```

Parameters

userName

The caller's user name
password

The caller's password

Returns

A newly created topic connection

Throws

CJMSException

If the JMS provider fails to create a topic connection due to some internal error.

CTopicPublisher

A client uses a CTopicPublisher object to publish messages on a topic. A CTopicPublisher object is the publish-subscribe form of a message producer.

Derives

CMessageProducer

getTopic

Gets the topic associated with this TopicPublisher.

Declaration

```
CTopic *getTopic() FMQCONST
                            throw (CJMSException *);
```

Returns

This publisher's topic

Throws

CJMSException

if the JMS provider fails to get the topic for this TopicPublisher due to some internal error.

publish

Publishes a message to the topic. Uses the TopicPublisher's default delivery mode, priority, and time to live.

Declaration

```
void publish(CMessage *msg)
                            throw (CJMSException *);
```

Parameters

message

The message to publish

Throws

CJMSException

If the JMS provider fails to publish the message due tosome internal error.

publish

Publishes a message to the topic, specifying delivery mode, priority, and time to live.

Declaration

```
void publish(CMessage *msg, mqint deliveryMode, mqint priority,

mqlong timeToLive)
                            throw (CJMSException *);
```

Parameters

message

The message to publish
deliveryMode

The delivery mode to use
Priority

The priority for this message
timeToLive

The message's lifetime (in milliseconds)

Throws

CJMSException

If the JMS provider fails to publish the message due to some internal error.

publish

Publishes a message to a topic for an unidentified message producer. Uses the TopicPublisher's default delivery mode, priority, and time to live.

Declaration

```
void publish(CTopic *topic, CMessage *msg)
                            throw (CJMSException *);
```

Parameters

topic

The topic to publish this message to
message

The message to publish.

Throws

CJMSException

If the JMS provider fails to publish the message due to some internal error.

publish

Publishes a message to a topic for an unidentified message producer, specifying delivery mode, priority and time to live. Typically, a message producer is assigned a topic at creation time; however, the JMS API also supports unidentified message producers, which require that the topic be supplied every time a message is published.

Declaration

```
void publish(CTopic *topic, CMessage *msg, mqint deliveryMode,

mqint priority, mqlong timeToLive)
                            throw (CJMSException *);
```

Parameters

topic

The topic to publish this message to
message

The message to publish
DeliveryMode

The delivery mode to use
priority

The priority for this message
timeToLive

The message's lifetime (in milliseconds)

Throws

CJMSException

If the JMS provider fails to publish the message due to some internal err

close

Closes the message producer. Since a provider may allocate some resources on behalf of a MessageProducer outside the Java virtual machine, clients should close them when they are not needed. Relying on garbage collection to eventually reclaim these resources may not be timely enough.

Declaration

```
void close()
                              throw (CJMSException *);
```

Throws

CJMSException

If the JMS provider fails to close the producer due to someinternal error.

getDeliveryMode

Gets the producer's default delivery mode.

Declaration

```
mqint getDeliveryMode() FMQCONST
                              throw (CJMSException *);
```

Returns

The message delivery mode for this message producer

Throws

CJMSException

If the JMS provider fails to get the delivery mode due to some internal error.

getDestination

Gets the destination associated with this MessageProducer.

Declaration

```
CDestination *getDestination() FMQCONST
                              throw (CJMSException *);
```

Returns

This producer's Destination

Throws

CJMSException

If the JMS provider fails to get the destination for this MessageProducer due to some internal error.

getDisableMessageID

Gets an indication of whether message IDs are disabled.

Declaration*

```
mqboolean getDisableMessageID() FMQCONST
                              throw (CJMSException *);
```

Returns

An indication of whether message IDs are disabled

Throws

CJMSException

If the JMS provider fails to determine if message IDs are disabled due to some internal error.

getDisableMesageTimestamp

Sets whether message timestamps are disabled.

Declaration

```
mqboolean getDisableMesageTimestamp() FMQCONST
                              throw (CJMSException *);
```

Parameters

value

Indicates if message timestamps are disabled

Throws

CJMSException

If the JMS provider fails to set timestamps to disabled due to some internal error.

getPriority

Gets the producer's default priority.

Declaration

```
mqint getPriority() FMQCONST
throw (CJMSException *);
```

Returns

The message priority for this message producer

Throws

CJMSException

If the JMS provider fails to get the priority due to some internal error.

getTimeToLive

Gets the default length of time in milliseconds from its dispatch time that a produced message should be retained by the message system.

Declaration

```
mqlong getTimeToLive() FMQCONST
                            throw (CJMSException *);
```

Returns

The message time to live in milliseconds; zero is unlimited

Throws

CJMSException

If the JMS provider fails to get the time to live due to some internal error.

send

Sends a message to a destination for an unidentified message producer. Uses the MessageProducer's default delivery mode, priority, and time to live.

Declaration

```
void send(CDestination *dest, CMessage *msg)
                            throw (CJMSException *);
```

Parameters

destination

The destination to send this message to
message

The message to send

Throws

CJMSException

If the JMS provider fails to send the message due to some internal error.

send

Sends a message to a destination for an unidentified message producer, specifying delivery mode, priority and time to live.

Declaration

```
void send(CDestination *dest, CMessage *msg, mqint deliveryMode,

mqint priority, mqint timeToLive)
                            throw (CJMSException *);
```

Parameters

destination

The destination to send this message to
message

The message to send
deliveryMode

The delivery mode to use
priority

The priority for this message
timeToLive

The message's lifetime (in milliseconds)

Throws

CJMSException

If the JMS provider fails to send the message due to some internal error.

send

Sends a message using the MessageProducer's default delivery mode, priority, and time to live.

Declaration

```
void send(CMessage *msg)
                              throw (CJMSException *);
```

Parameters

message

The message to send

Throws

CJMSException

If the JMS provider fails to send the message due to some internal error.

send

Sends a message to the destination, specifying delivery mode, priority, and time to live.

Declaration

```
void send(CMessage *msg, mqint deliveryMode, mqint priority,

mqlprong timeToLive)
                              throw (CJMSException *);
```

Parameters

message

The message to send
deliveryMode

the delivery mode to use
priority

the priority for this message
timeToLive

the message's lifetime (in milliseconds)

Throws

CJMSException

If the JMS provider fails to send the message due to some internal error.

setDeliveryMode

Sets the producer's default delivery mode.Delivery mode is set to PERSISTENT by default.

Declaration

```
void setDeliveryMode(mqint deliveryMode)
                            throw (CJMSException *);
```

Parameters

deliveryMode

The message delivery mode for this message producer; legal values are DeliveryMode.NON_PERSISTENT and DeliveryMode.PERSISTENT

Throws

CJMSException

If the JMS provider fails to set the delivery mode due to some internal error.

setDisableMessageID

Sets whether message IDs are disabled.

Declaration

```
void setDisableMessageID(mqboolean value)
                            throw (CJMSException *);
```

Parameters

value

Indicates if message IDs are disabled

Throws

CJMSException

If the JMS provider fails to set message ID to disabled due to some internal error.

setDisableMessageTimestamp

Sets whether message timestamps are disabled.

Declaration

```
void setDisableMessageTimestamp(mqboolean value)
                            throw (CJMSException *);
```

Parameters

value

Indicates if message timestamps are disabled

Throws

CJMSException

If the JMS provider fails to set timestamps to disabled due to some internal error.

setPriority

Sets the producer's default priority. The JMS API defines ten levels of priority value, with 0 as the lowest priority and 9 as the highest.Priority is set to 4 by default.

Declaration

```
void setPriority(mqint defaultPriority)
                            throw (CJMSException *);
```

Parameters

defaultPriority

The message priority for this message producer; must be a value between 0 and 9

Throws

CJMSException

If the JMS provider fails to set the priority due to some internal error.

setTimeToLive

Sets the default length of time in milliseconds from its dispatch time that a produced message should be retained by the message system. Time to live is set to zero by default.

Declaration

```
void setTimeToLive(mqlong timeToLive)
                            throw (CJMSException *);
```

Parameters

timeToLive

The message time to live in milliseconds; zero is unlimited

Throws

CJMSException

If the JMS provider fails to set the time to live due to some internal error.

CTopicRequestor

The CTopicRequestor helper class simplifies making service requests.

close

Declaration

```
void close()
                              throw (CJMSException *);
```

Closes the TopicRequestor and its session.

Throws

CJMSException

If the JMS provider fails to close the TopicRequestor due to some internal error.

request

Sends a request and waits for a reply. The temporary topic is used for the JMSReplyTo destination; the first reply is returned, and any following replies are discarded.

Declaration

```
CMessage *request(CMessage *msg)
                              throw (CJMSException *);
```

Parameters

message

The message to send

Returns

The reply message

Throws

CJMSException

If the JMS provider fails to complete the request due to some internal error.

request

Sends a request and waits for a reply until timeout. The temporary topic is used for the JMSReplyTo destination; the first reply is returned, and any following replies are discarded.

Declaration

```
CMessage *request(CMessage *msg, mqlong timeout)
                              throw (CJMSException *);
```

Parameters

message

The message to send

timeout

Time until which to wait for message

Returns

The reply message

Throws

CJMSException

If the JMS provider fails to complete the request due to some internal error.

CTopicSession

A CTopicSession object provides methods for creating CTopicPublisher, CTopicSubscriber, and CTemporaryTopic objects. It also provides a method for deleting its client's durable subscribers.

Derives

CSession

createPublisher

Creates a publisher for the specified topic. A client uses a TopicPublisher object to publish messages on a topic. Each time a client creates a TopicPublisher on a topic, it defines a new sequence of messages that have no ordering relationship with the messages it has previously sent.

Declaration

```
CTopicPublisher *createPublisher(CTopic *topic)
                            throw (CJMSException *);
```

Parameters

topic

The Topic to publish to, or null if this is an unidentified producer

Throws

CJMSException

If the session fails to create a publisher due to someinternal error.

createSubscriber

Creates a nondurable subscriber to the specified topic. A client uses a TopicSubscriber object to receive messages that have been published to a topic. Regular TopicSubscriber objects are not durable. They receive only messages that are published while they are active.

Declaration

```
CTopicSubscriber *createSubscriber(CTopic *topic)
```

```
                              throw (CJMSException *);
```

Parameters

topic

The Topic to subscribe to

Throws

CJMSException

If the session fails to create a subscriber due to some internal error.

InvalidDestinationException

If an invalid topic is specified.

createSubscriber

Creates a nondurable subscriber to the specified topic.

Declaration

```
CTopicSubscriber *createSubscriber(CTopic *topic,

mqcstring messageSelector, mqboolean noLocal)
                              throw (CJMSException *);
```

Parameters

topic

The Topic to subscribe to
messageSelector

Only messages with properties matching the message selector expression are delivered. A value of null or an empty mqcstring indicates that there is no message selector for the message consumer.
noLocal

If set, inhibits the delivery of messages published by its own connection

Throws

CJMSException

If the session fails to create a subscriber due to some internal error.

close

Closes the session.

Declaration

```
void close()
                              throw (CJMSException *);
```

Throws

CJMSException

If the JMS provider fails to close the session due to some internal error.

commit

Commits all messages done in this transaction and releases any locks currently held.

Declaration

```
void commit()
    throw (CJMSException *);
```

Throws

CJMSException

If the JMS provider fails to commit the transaction due to some internal error.

createBytesMessage

Creates a BytesMessage object. A BytesMessage object is used to send a message containing a stream of uninterpreted bytes.

Declaration

```
CBytesMessage *createBytesMesage()
                            throw (CJMSException *);
```

Throws

CJMSException

If the JMS provider fails to create this message due to some internal error.

createDurableSubscriber

Creates a durable subscriber to the specified topic.

Declaration

```
CTopicSubscriber *createDurableSubscriber(CTopic *topic, mqcstring name)
                            throw (CJMSException *);
```

Specified by

createDurableSubscriber in interface Session

Parameters

topic

The non-temporary Topic to subscribe to

name

The name used to identify this subscription

Throws

CJMSException

If the session fails to create a subscriber due to some internal error.

createDurableSubscriber

Creates a durable subscriber to the specified topic.

Specified by

createDurableSubscriber in interface Session

Declaration

```
CTopicSubscriber *createDurableSubscriber(CTopic *topic, mqcstring name,mqcstring
messageSelector, mqboolean noLocal)
                            throw (CJMSException *);
```

Parameters

topic

The non-temporary Topic to subscribe to
name

The name used to identify this subscription
messageSelector

Only messages with properties matching the message selector expression are delivered. A value of null or an empty mqcstring indicates that there is no message selector for the message consumer.
noLocal

If set, inhibits the delivery of messages published by its own connection

Throws

CJMSException

If the session fails to create a subscriber due to some internal error.

createMapMessage

Creates a MapMessage object. A MapMessage object is used to send a self-defining set of name-value pairs, where names are String objects and values are primitive values in the Java programming language.

Declaration

```
CMapMessage *createMapMessage()
                            throw (CJMSException *);
```

Throws

CJMSException

If the JMS provider fails to create this message due to some internal error.

createStreamMessage

Creates a StreamMessage object. A StreamMessage object is used to send a self-defining stream of primitive values in the Java programming language.

Declaration

```
CStreamMessage *createStreamMessage()
                            throw (CJMSException *);
```

Throws

CJMSException

If the JMS provider fails to create this message due tosome internal error.

createTemporaryQueue

Creates a TemporaryQueue object. Its lifetime will be that of the Connection unless it is deleted earlier.

Declaration

```
CTemporaryQueue *createTemporaryQueue()
                            throw (CJMSException *);
```

Returns

A temporary queue identity

Throws

CJMSException

If the session fails to create a temporary queue due to some internal error.

createTextMessage

Creates a TextMessage object. A TextMessage object is used to send a message containing a String object.

Declaration

```
CTextMessage *createTextMessage()
                            throw (CJMSException *);
```

Throws

CJMSException

If the JMS provider fails to create this message due tosome internal error.

createTextMessage

Creates an initialized TextMessage object. A TextMessage object is used to send a message containing a String.

Declaration

```
CTextMessage *createTextMessage(mqcstring text)
                            throw (CJMSException *);
```

Parameters

text

The mqcstring used to initialize this message

Throws

CJMSException

If the JMS provider fails to create this message due to some internal error.

createTopic

Creates a topic identity given a Topic name.

Declaration

```
CTopic *createTopic(mqcstring topicName)
                            throw (CJMSException *);
```

Parameters

topicName

The name of this Topic
Returns

A Topic with the given name

Throws

CJMSException

If the session fails to create a topic due to some internal error.

getMessageListener

Returns the session's distinguished message listener (optional).

Declaration

```
CMessageListener *getMessageListener()
```

```
                                throw (CJMSException *);
```

Returns

The message listener associated with this session

Throws

CJMSException

If the JMS provider fails to get the message listener due to an internal error.

recover

Stops message delivery in this session, and restarts message delivery with the oldest unacknowledged message.

Declaration

```
void recover()
                                throw (CJMSException *);
```

Throws

CJMSException

If the JMS provider fails to stop and restart message delivery due to some internal error.

rollback

Rolls back any messages done in this transaction and releases any locks currently held.

Declaration

```
void rollback()
                                throw (CJMSException *);
```

Throws

CJMSException

If the JMS provider fails to roll back the transaction due to some internal error.

setMessageListener

Sets the session's distinguished message listener.

Declaration

```
void setMessageListener(CMessageListener *messageListener)
                                throw (CJMSException *);
```

Parameters

listener

The message listener to associate with this session

Throws

CJMSException

If the JMS provider fails to set the message listener due to an internal error.

unsubscribe

Unsubscribes a durable subscription that has been created by a client.

Declaration

```
void unsubscribe(mqcstring name)
                            throw (CJMSException *);
```

Parameters

name

The name used to identify this subscription

Throws

CJMSException

If the session fails to unsubscribe to the durable subscription due to some internal error.

CTopicSubscriber

A client uses a CTopicSubscriber object to receive messages that have been published to a topic. A CTopicSubscriber object is the publish/subscribe form of a message consumer.

Derives

CMessageConsumer

close

Closes the message consumer.

Declaration

```
void close()
                            throw (CJMSException *);
```

Throws

CJMSException

If the JMS provider fails to close the consumer due to some internal error.

getMessageListener

Gets the message consumer's MessageListener.

Declaration

```
CMessageListener *getMessageListener() FMQCONST
                              throw (CJMSException *);
```

Returns

The listener for the message consumer, or null if no listener is set

Throws

CJMSException

If the JMS provider fails to get the message listener due to some internal error.

getMessageSelector

Gets this message consumer's message selector expression.

Declaration

```
mqcstring getMessageSelector() FMQCONST
                              throw (CJMSException *);
```

Returns

This message consumer's message selector, or null if no message selector exists for the message consumer (that is, if the message selector was not set or was set to null or the empty mqcstring).

Throws

CJMSException

If the JMS provider fails to get the message selector due to some internal error.

receive

Receives the next message produced for this message consumer. This call blocks indefinitely until a message is produced or until this message consumer is closed. If this receive is done within a transaction, the consumer retains the message until the transaction commits.

Declaration

```
CMessage *receive()
                              throw (CJMSException *);
```

Returns

The next message produced for this message consumer, or null if this message consumer is concurrently closed

Throws

CJMSException

If the JMS provider fails to receive the next message due to some internal error.

receive

Receives the next message that arrives within the specified timeout interval. This call blocks until a message arrives, the timeout expires, or this message consumer is closed. A timeout of zero never expires, and the call blocks indefinitely.

Declaration

```
CMessage *receive(mqlong timeout)
                            throw (CJMSException *);
```

Parameters

timeout

The timeout value (in milliseconds)

Returns

The next message produced for this message consumer, or null if the timeout expires or this message consumer is concurrently closed

Throws

CJMSException

If the JMS provider fails to receive the next message due to some internal error.

receiveNoWait

Receives the next message if one is immediately available.

Declaration

```
CMessage *receiveNoWait()
                            throw (CJMSException *);
```

Returns

The next message produced for this message consumer, or null if one is notavailable

Throws

CJMSException

If the JMS provider fails to receive the next message due to some internal error.

setMessageListener

Sets the message consumer's MessageListener.

Declaration

```
void setMessageListener(CMessageListener *messageListener)
                            throw (CJMSException *);
```

Parameters

listener

The listener to which the messages are to be delivered

Throws

CJMSException

If the JMS provider fails to set the message listener due to some internal error.

getNoLocal

Gets the NoLocal attribute for this subscriber. The default value for thisattribute is false.

Declaration

```
mqboolean getNoLocal() FMQCONST
                            throw (CJMSException *);
```

Returns

TRUE if locally published messages are being inhibited

Throws

CJMSException

If the JMS provider fails to get the NoLocal attribute for this topic subscriber due to some internal error.

getTopic

Gets the Topic associated with this subscriber.

Declaration

```
CTopic *getTopic() FMQCONST
                            throw (CJMSException *);
```

Returns

This subscriber's Topic

Throws

CJMSException

If the JMS provider fails to get the topic for this topic subscriber due to some internal error.

CLogHandler

This class is used for logging to a file. It is a singleton class and only one instance of Logger exists for the whole application.

The functions in the class CLogHandler are:

setLoggerName

Sets file name[in which data is logged] to name. The default file name is cclient.

Declaration: static mqboolean setLoggerName(mqstring name)

throw (CJMSException*);

For example, CLogHandler::setLoggerName("Publisher");

getLogHandler

This function gets the reference to the logHandler, using which data can be logged from application.

Declaration: static CLogHandler* getLogHandler()

throw (CJMSException*);

setTraceLevel

The trace level can also be set using this function. The legal values for the "level" are:

1. "ERROR"
2. "INFO"
3. "DEBUG"

Declaration: mqboolean setTraceLevel(mqstring level);

logData

It logs the given data to the log file. This function is variable argument function. The usage of this function is similar to printf function.

Declaration: mqboolean logData(mqstring info, ...)

throw (CJMSException*);

Example

```
CLogHandler *logger = CLogHandler::getLogHandler();
```

logger->logData("Logging from file - %s, func -%s", "FileName", "funcName");

In log file it is printed as follows:

```
Fri Dec 19 14:41:25 2008  :APPL: Logging from file - FileName, func –funcName
```

CCSPManager

This class is used to create CSPBrowser, which is used for browsing messages stored in CSP cache.

Constructor

Constructors are used for creating CCSPManager class. The path to the CSP has to be given as parameter to the constructor.

Declaration: CCSPManager(mqcstring cspPath);

createCSPBrowser

Creates CCSPBrowser for the location specified for the manager.

Declaration: CCSPBrowser* createCSPBrowser() throw (CJMSException *);

CCSPBrowser

CSP Browser is used to browse the messages stored in Client-Side Persistent Cache. While browsing messages are read from CSP and are not deleted. So, next time when the Server is re-connected, messages are sent to the Server.

The classes used for CSP Browsing are CCSPBrowser and CCSPEnumeration.

The following functions are used for CSP Browsing.

getAllConnections

This function returns the enumeration of all the Connection IDs whose messages are stored in CSP

Declaration: CEnumeration* getAllConnections()

throw (CJMSException *);

getTopicsForConnection

This function returns the Enumeration of all the topic names for the specified connection with clientID as connectionID.

Declaration: CEnumeration* getTopicsForConnection(mqcstring connectionID)

throw (CJMSException *);

getQueuesForConnection

This function returns the Enumeration of all the queue names for the specified connection with clientID as connectionID.

Declaration: CEnumeration* getQueuesForConnection(mqcstring connectionID)

throw (CJMSException *);

browseMessagesOnQueue

This function returns the enumeration of messages stored in the queue specified by queueName. It searches for messages in queue in all the existing connections. If **checkTransacted** is TRUE, transacted messages are also browsed.

Declaration: CCSPEnumeration* browseMessagesOnQueue(mqcstring queueName, mqboolean checkTransacted)

throw (CJMSException *);

browseMessagesOnQueue

This function returns the enumeration of messages stored in the queue specified by queueName. It searches for messages in queue for the connection specified by connectionID. If **checkTransacted** is TRUE, transacted messages are also browsed.

Declaration: CCSPEnumeration* browseMessagesOnQueue(mqcstring connectionID, mqcstring queueName, mqboolean checkTransacted)

throw (CJMSException *);

browseMessagesOnTopic

This function returns the enumeration of messages stored in the topic specified by topicName. It searches for messages in topic in all the existing connections. If **checkTransacted** is TRUE, transacted messages are also browsed.

Declaration: CCSPEnumeration* browseMessagesOnTopic(mqcstring topicName, mqboolean checkTransacted)

throw (CJMSException *);

browseMessagesOnTopic

This function returns the enumeration of messages stored in the topic specified by topicName. It searches for messages in topic for the connection specified by connectionID. If **checkTransacted** is TRUE, transacted messages are also browsed.

Declaration: CCSPEnumeration* browseMessagesOnTopic(mqcstring connectionID, mqcstring topicName, mqboolean checkTransacted)

throw (CJMSException *);

numberOfMessagesinQueue

This function returns the number of messages stored in the queue specified by queueName for a given connection with clientID connID.

If **checkTransacted** is TRUE, transacted messages are also counted.

Declaration: mqlong numberOfMessagesinQueue(mqcstring connID, mqcstring queueName, mqboolean checkTransacted)

throw (CJMSException *);

numberOfMessagesinTopic

This function returns the number of messages stored in the topic specified by topicName for a given connection with clientID connID.

If **checkTransacted** is TRUE, transacted messages are also counted.

Declaration: mqlong numberOfMessagesinTopic(mqcstring connID, mqcstring topicName, mqboolean checkTransacted)

throw (CJMSException *);

CCSPEnumeration

This is the enumeration class that gets the list of messages from CSP. The functions used in this class,

hasMoreElements

This function checks whether more elements are available in the enumeration.

Declaration: mqboolean hasMoreElements()

throw (CJMSException *);

nextElement

This function returns the next element of this enumeration if this enumeration object has at least one more element to provide.

Declaration: CMessage* nextElement()

throw (CJMSException *);

Large Message Support

With Large Message Support (LMS) in FioranoMQ, clients can transfer large files in the form of large messages with theoretically no limit on the message size. Large messages can be attached with any JMS message and the client can be sure of a reliable and secure transfer of the message through FioranoMQ server. Please refer to chapter 15 of **FioranoMQ Concepts Guide** for detailed explanation of LMS.

The following classes and functions are used in LMS.

CfioranoConnection

These two functions are used for resuming any pending large messages.

getUnfinishedMessagesToSend

This function Returns the enumeration of Messages whose transfers are pending as they were not fully sent.

Declaration: CRecoverableMessagesEnum *getUnfinishedMessagesToSend(FioranoConnection fc);

getUnfinishedMessagesToReceive

This function Returns the enumeration of Messages which were not fully received and need to be resumed.

Declaration: CRecoverableMessagesEnum *getUnfinishedMessagesToReceive(FioranoConnection fc);

CRecoverableMessagesEnum

This class is the enumeration of Unfinished Messages.

hasMoreElements

Declaration: bool hasMoreElements();

Returns true if there are more elements in the enumeration, false otherwise

nextElement

Declaration: CMessage *nextElement(RMEnum enumr);

return the next element (CMessage Object) of the enumeration.

ClargeMessage

getMessageStatus

Declaration: CLMTransferStatus *getMessageStatus();

Gets the status of the message. Status includes number of bytes transferred, number of bytes left to be transferred, etc.

setLMStatusListener

Declaration: void setLMStatusListener(CLMStatusListener *listener, int updateFrequency);

Sets the status listener for the message. This function is used to know the status of message transfer asynchronously.

getLMStatusListener

Declaration: CLMStatusListener *getLMStatusListener()

Gets the status listener for the message.

saveTo

Declaration: void saveTo(mqcstring fileName, bool isBlocking);

Saves the contents of the message in the file specified.

resumeSaveTo

Declaration: void resumeSaveTo(bool isBlocking);

Resumes saving the contents of the message in the file specified.

resumeSend

Declaration: void resumeSend();

Resumes an incomplete transfer. This function is used to resume a message transfer which could not be completed earlier either due to some internal error or due to some problem at the client side.

cancelAllTransfers

Declaration: void cancelAllTransfers();

Cancels all message transfers which are currently transferring this message.

A cancelled transfer also removes the resume information of the transfer. Hence a transfer once cancelled cannot be resumed.

cancelTransfer

Declaration: void cancelTransfer(int consumerID);

Cancels the transfer specified by the consumerID.

A cancelled transfer also removes the resume information of the transfer. Hence a transfer once cancelled cannot be resumed.

suspendAllTransfers

Declaration: void suspendAllTransfers ();

Suspends all the message transfers which are transferring this large message temporarily.

Suspending a transfer only stops the thread which is doing the message transfer and does not delete resume related information. Hence, a suspended transfer can be resumed using resume functions

suspendTransfer

Declaration: void suspendTransfer(int consumerID);

Suspends the transfer specified by the consumerID temporarily. Suspending a transfer only stops the thread which is doing the message transfer and does not delete resume related information. Hence, a stopped transfer can be resumed using resume functions

setFragmentSize

Declaration: void setFragmentSize(int size);

Sets the fragment size for the message.

getFragmentSize

Declaration: int getFragmentSize();

gets the fragment size of the message.

setWindowSize

Declaration: void setWindowSize(int size);

Sets the frequency after which acknowledgement will be sent

getWindowSize

Declaration: int getWindowSize();

gets the window size of the message.

setRequestTimeoutInterval

Declaration: void setRequestTimeoutInterval(long timeout);

Sets the time until which the sender will wait for message transfer to start

getRequestTimeoutInterval

Declaration: long getRequestTimeoutInterval();

Gets the time until which the sender will wait for message transfer to start

setResponseTimeoutInterval

Declaration: void setResponseTimeoutInterval(long responseInterval);

Sets the time until which the sender/receiver will wait for message from the other end.

getResponseTimeoutInterval

Declaration: long getResponseTimeoutInterval();

Gets the time until which the sender/receiver will wait for messages from the other end.

CLMStatusListener

This class listens to the status of the Large Message received or sent. This has one virtual function which has to over ridden in Status Listener implementation. Refer to the lms samples for the sample implementation of this class

onLMStatus

Declaration: void onLMStatus(CLMTransferStatus *status, bool exception);

CLMTransferStatus

The following functions are to get the status information from CLMTransferStatus class. This class includes status of the Message transfer such as number of bytes transferred, number of bytes left to be transferred, etc.

getBytesTransferred

Declaration: mqlong getBytesTransferred();

Returns the number of bytes transferred.

getBytesToTransfer

Declaration: mqlong getBytesToTransfer();

Returns the number of bytes to be transferred.

getLastFragmentID

Declaration: mqlong getLastFragmentID();

Returns the ID of the last fragment.

getPercentageProgress

Declaration: float getPercentageProgress();

Returns the percentage of the progress of message transfer.

getStatus

Declaration: mqbyte getStatus();

Returns the status of the Message transfer.

LM_TRANSFER_NOT_INIT or 1

Indicates that the transfer has not yet started

LM_TRANSFER_IN_PROGRESS or 2

Indicates that transfer is currently in progress

LM_TRANSFER_DONE or 3

Indicates that the transfer is complete for one consumer

LM_TRANSFER_ERR or 4

Indicates that an error occurred during the transfer

LM_ALL_TRANSFER_DONE or 5

Indicates that the transfer is complete for all consumers

isTransferComplete

Declaration: bool isTransferComplete();

Returns true if transfer completes, else returns false.

isTransferCompleteForAll

Declaration: bool isTransferCompleteForAll();

Returns true if all the transfers are completed, else returns false.

getLargeMessage

Declaration: Message getLargeMessage();

Returns the large message for which this status is created.

getConsumerID

Declaration: int getConsumerID();

Returns the consumerID of the connection that is being used.

Administration API

CAdminConnectionFactory

A client uses a CAdminConnectionFactory object to create CAdminConnection

objects with a JMS provider.

Derives

CConnectionFactory

createAdminConnectionDefParams

Creates a admin connection with the default user identity.

Declaration

```
CAdminConnection *createAdminConnectionDefParams()
                        throw (CJMSException *);
```

Returns

A newly created admin connection

Throws

CJMSException

If the JMS provider fails to create a admin connection due to some internal error.

createAdminConnection

Creates a admin connection with specified user identity.

Declaration

```
CAdminConnection *createAdminConnection(mqcstring username, mqcstring password)
                        throw (CJMSException *);
```

Parameters

username

The caller's user name
Password

The caller's password

Returns

A newly created admin connection

Throws

CJMSException

If the JMS provider fails to create a admin connection due to some internal error.

CAdminConnection

AdminConnectionFactories are used to create AdminConnections with the FioranoMQ server. The AdminConnection is created with the server running on the ConnectURL specified in the AdminConnectionFactory and if the same is unavailable then the RTL tries to make a connection with a BackupURL, if any. AdminConnections can be created using default user identity ("admin","passwd" in case of FioranoMQ) or by specifying a username and password.

getMQAdminService

A MQAdminService object provides methods for creating and deleting Queues, Topics, QueueConnectionFactory and TopicConnectionFactory objects. Various get/set methods specify the object properties to and from the server.

Declaration

```
CMQAdminService* getMQAdminService()
                        throw (CJMSException*);
```

Returns

A newly created admin connection

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

Close

Closes the connection.

Declaration

```
void close()
throw (CJMSException *);
```

Returns

void

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

CMQAdminService

getNumberOfActiveClientConnections

Gets the number of active client connections to the MQ Server.

Declaration

```
mqint getNumberOfActiveClientConnections()
                        throw (CJMSException*);
```

Returns

mqint value representing the number of active clients connected to the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getDurableSubscribersForTopic

Returns a pointer to enumeration of all the subscriber names on the specified topic.

Declaration

```
CEnumeration* getDurableSubscribersForTopic(mqcstring topicName)
                        throw (CJMSException*);
```

Parameters

topicName

The topic name.

Returns

Pointer to CEnumeration object of all the subscriber names on the specified topic.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getClientIDs

Returns a pointer to enumeration of all client ids.

Declaration

```
CEnumeration* getClientIDs()
                        throw (CJMSException*);
```

Returns

Pointer to CEnumeration object of all the client ids.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getPTPClientIDs

Returns a pointer to enumeration of all client ids for PTP (Point to Point JMS model).

Declaration

```
CEnumeration* getPTPClientIDs()
                        throw (CJMSException*);
```

Returns

Pointer to CEnumeration object of all the client ids on PTP.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getPubSubClientIDs

Returns a pointer to enumeration of all client ids for PubSub (Publish / Subscribe JMS model).

Declaration

```
CEnumeration* getPubSubClientIDs()
```

```
                        throw (CJMSException*);
```

Returns

Pointer to CEnumeration object of all the client ids on PubSub.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getSubscriberIDs

Returns a pointer to enumeration of all subscriber ids on the specified client id.

Declaration

```
CEnumeration* getSubscriberIDs(mqcstring clientID)
throw (CJMSException*);
```

Parameters

clientID

String representing the client id.

Returns

Pointer to CEnumeration object of all the Subscriber ids on the specified client id.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getSubscriptionTopicName

Returns a mqcstring subscription topic name for the specified clientID and subscriberID combo.

Declaration

```
mqcstring getSubscriptionTopicName(mqcstring clientID, mqcstring subscriberID)
                        throw (CJMSException*);
```

Parameters

clientID

String representing the client id
subscriberID

String representing the subscriber id.

Returns

mqcstring subscription topic name for the specified clientID and subscriberID combo.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getNumberOfDeliverableMessages1

Returns a mqlong object for the number of deliverable messages on the specified queue.

Declaration

```
mqlong getNumberOfDeliverableMessages1(mqcstring queueName)
throw (CJMSException*);
```

Parameters

queueName

String representing the queue name.

Returns

mqlong object for the number of deliverable messages on the specified queue.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getNumberOfDeliverableMessages2

Returns a mqlong object for the number of deliverable messages on the specified clientID and subscriberID combo.

Declaration

```
mqlong getNumberOfDeliverableMessages2(mqcstring clientID, mqcstring subscriberID)
throw (CJMSException*);
```

Parameters

clientID

String representing the client id.
subscriberID

String representing the subscriber id.

Returns

mqlong object for the number of deliverable messages on the specified clientID and subscriberID combo.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getNumberOfUndeletedMessages

Returns a mqlong object for the number of undeleted messages on the server for the specified queue name.

Declaration

```
mqlong getNumberOfUndeletedMessages(mqcstring queueName)
throw (CJMSException*);
```

Parameters

queueName

String representing the queue Name.

Returns

mqlong object for the number of undeleted messages on the server for the specified queue name.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

Unsubscribe

Unsubcribes a durable subscriber for the specified clientID and subscriberID combo.

Declaration

```
mqboolean unsubscribe(mqcstring clientID, mqcstring subscriberID)
throw (CJMSException*);
```

Parameters

clientID

String representing the client id.
subscriberID

String representing the subscriber id.

Returns

mqboolean when unsubscribing a durable subscriber for the specified clientID and subscriberID combo.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

purgeSubscriptionMessages

Purge all messages for a durable subscriber for the specified clientID and subscriberID combo.

Declaration

```
mqboolean purgeSubscriptionMessages(mqcstring clientID, mqcstring subscriberID)throw
(CJMSException*);
```

Parameters

clientID

String representing the client id.
subscriberID

String representing the subscriber id.

Returns

mqboolean when purging all messages for a durable subscriber for the specified clientID and subscriberID combo.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error

createTopic

Create a topic with the specified topic meta data on the MQ Server.

Declaration

```
mqboolean createTopic(CTopicMetaData* topicMetaData)
throw (CJMSException*);
```

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

createQueue

Create a queue with the specified queue meta data on the MQ Server.

Declaration

```
mqboolean createQueue(CQueueMetaData* queueMetaData)
throw (CJMSException*);
```

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

deleteTopic

Delete a topic with the specified topic name.

Declaration

```
mqboolean deleteTopic(mqcstring topicName)
throw (CJMSException*);
```

Parameters

topicName

String representing the topic name.

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

deleteQueue

Delete a queue with the specified queue name.

Declaration

```
mqboolean deleteQueue(mqcstring queueName)
throw (CJMSException*);
```

Parameters

queueName

String representing the queue name.

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

createTopicConnectionFactory

Create a topicConnectionFactory with the specified topicConnectionFactory meta data on the MQ Server.

Declaration

```
mqboolean createTopicConnectionFactory( CTopicConnectionFactoryMetaData *tcfMetaData)
throw (CJMSException*);
```

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

createQueueConnectionFactory

Create a queueConnectionFactory with the specified queueConnectionFactory meta data on the MQ Server.

Declaration

```
mqboolean createQueueConnectionFactory( CQueueConnectionFactoryMetaData *qcfMetaData)
throw (CJMSException*);;
```

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

deleteQueueConnectionFactory

Delete a queueConnectionFactory with the specified queueConnectionFactory name.

Declaration

```
mqboolean deleteQueueConnectionFactory(mqcstring qcfName)
throw (CJMSException*);
```

Parameters

qcfName

String representing the queue connection factory name.

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

deleteTopicConnectionFactory

Delete a topicConnectionFactory with the specified topicConnectionFactory name.

Declaration

```
mqboolean deleteTopicConnectionFactory(mqcstring tcfName)
throw (CJMSException*);
```

Parameters

tcfName

String representing the topic connection factory name.

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getCurrentUsers

Returns a pointer to enumeration of all the current users on the MQ Server.

Declaration

```
CEnumeration* getCurrentUsers()
throw (CJMSException*);
```

Returns

Pointer to CEnumeration object of all the current users on the MQ Server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

restartServer

Restarts the MQ Server.

Declaration

```
mqboolean restartServer()
throw (CJMSException*);
```

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

shutdownServer

Shuts down the MQ Server.

Declaration

```
mqboolean shutdownServer()
throw (CJMSException*);
```

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

shutDownActiveHAServer

Shuts down the Active HA MQ Server.

Declaration

```
mqboolean shutDownActiveHAServer()
throw (CJMSException*);
```

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

shutDownPassiveHAServer

Shuts down the Passive HA MQ Server.

Declaration

```
mqboolean shutDownPassiveHAServer()
throw (CJMSException*);
```

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

purgeQueueMessages1

Purges all the messages on the specified queue.

Declaration

```
mqboolean purgeQueueMessages1(mqcstring queueName)
throw (CJMSException*);
```

Parameters

queueName

String representing the queue name.

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

purgeQueueMessages2

Purges all the messages on the specified queue forcefully.

Declaration

```
mqboolean purgeQueueMessages2(mqcstring queueName, mqboolean forcefully)
throw (CJMSException*);
```

Parameters

queueName

String representing the queue name.
forcefully

mqboolean representing forcefully purge or not.

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

showStatusOfAllQueues

Shows the status of all the queues on the server.

Declaration

```
mqboolean showStatusOfAllQueues()
throw (CJMSException*);
```

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

deleteMessagesOnServer

Delete messages on the specified queue with startindex upto the endindex and with specified priority.

Declaration

```
mqint deleteMessagesOnServer(mqcstring queueName, mqlong startIndex, mqlong endIndex, mqint
priority)
throw (CJMSException*);
```

Parameters

queueName

String representing the queue name.
startIndex

mqlong start index.
endIndex

mqlong end index.
Priority

Mqint message proiority.

Returns

Mqint representing the number of messages deleted on the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

loadAdminObjects

Load admin objects on the server.

Declaration

```
mqboolean loadAdminObjects()
throw (CJMSException*);
```

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

CTopicMetaData

CTopicMetaData object represents the topic object inside the MQServer.

Various get/set methods are used specify and retrieve object properties.

setName

Sets the topic name in the topicMetaData object.

Declaration

```
mqboolean setName(mqcstring metaDataName)
throw (CJMSException *);
```

Parameters

metaDataName

String representing the topic name.

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

setDescription

Sets the topic description in the topicMetaData object.

Declaration

```
mqboolean setDescription(mqcstring metaDataDescription)
throw (CJMSException *);
```

Parameters

metaDataDescription

String representing the topic description.

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getName

Gets the name of the topic from the topicMetaData object.

Declaration

```
mqcstring getName()
throw (CJMSException *);
```

Returns

mqcstring value representing the topic name, NULL otherwise.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getDescription

Gets the description of the topic from the topicMetaData object.

Declaration

```
mqcstring getDescription()
throw (CJMSException *);
```

Returns

mqcstring value representing the topic description, NULL otherwise.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

CQueueMetaData

CQueueMetaData object represents the queue object inside the MQServer.

Various get/set methods are used specify and retrieve object properties.

setName

Sets the queue name in the queueMetaData object.

Declaration

```
mqboolean setName(mqcstring metaDataName)
throw (CJMSException *);
```

Parameters

metaDataName

String representing the queue name.

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

setDescription

Sets the queue description in the queueMetaData object.

Declaration

```
mqboolean setDescription(mqcstring metaDataDescription)
hrow (CJMSException *);
```

Parameters

metaDataDescription

String representing the topic description.

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getName

Gets the name of the queue from the queueMetaData object.

Declaration

```
mqcstring getName()
throw (CJMSException *);
```

Returns

mqcstring value representing the queue name, NULL otherwise.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getDescription

Gets the description of the queue from the queueMetaData object.

Declaration

```
mqcstring getDescription()
throw (CJMSException *);
```

Returns

mqcstring value representing the queue description, NULL otherwise.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

CTopicConnectionFactoryMetaData

Class CTopicConnectionFactoryMetaData represents the MetaData information for a CTopicConnectionFactory at it is stored in an LDAP directory. Objects of this class are used to create CTopicConnectionFactory objects.

allowAutoRevalidation

Sets the value for auto-revalidation. Setting this to true revalidates all the connections automatically whenever connection with the server goes down. This method does not allow one to set durable connections to true, i.e., no local caching of messages is possible using this API.

Declaration

```
mqboolean allowAutoRevalidation(mqcstring allowAutoRevaidation)
throw (CJMSException *);
```

Parameters

allowAutoRevaidation

mqcstring to indicate allowAutoRevalidation setting as TRUE or FALSE.

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

allowDurableConnections

Enables the durable connections for this ConnectionFactory. This means that all connections created using this ConnectionFactory will be durable connections if this support is enabled in the FioranoMQ Server.

Declaration

mqboolean allowDurableConnections(mqboolean allowDurableConnections);

Parameters

allowDurableConnections

mqboolean to indicate allowDurableConnections setting as TRUE or FALSE.

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

areDurableConnectionsAllowed

Gives the status whether durable connection are allowed or not for this connection factory.

Declaration

```
mqboolean areDurableConnectionsAllowed();
```

Returns

mqboolean value for the status whether durable connection are allowed or not for this connection factory.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

compareConnectURLs

Compares the ConnectURLS in the CTopicConnectionFactoryMetaData.

Declaration

```
mqboolean compareConnectURLs(CTopicConnectionFactoryMetaData* metaData);
```

Parameters

metadata

CTopicConnectionFactoryMetaData pointer.

Returns

mqboolean value to indicate if the urls are equal or not equal.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

disableCSPStoredMessageSend

Disables the sending of any pending messages in the client side store of all durable connections created using this ConnectionFactory. If no value is set then send of pending messages is enabled.

Declaration

```
mqboolean disableCSPStoredMessageSend(mqcstring dontSend);
```

Parameters

dontSend

mqcstring as TRUE or FALSE.

Returns

mqboolean value to indicate if the urls are equal or not equal.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

disablePing

Disables the ping functionality for this CTopicConnectionFactory. This would mean that no pinging would be done by the FioranoMQ runtime for all connections created using this CTopicConnectionFactory even if pinging is enabled in the FioranoMQ server.

Declaration

```
mqboolean disablePing(mqboolean ping);
```

Parameters

Ping

mqboolean as TRUE or FALSE.

Returns

mqboolean value to indicate success or failure.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

disableReaderCache

Disables the reader cache in the connection buffers. The memory will be re-allocated for each new read sequence on each of the connection created out of this connection factory.

Declaration

```
mqboolean disableReaderCache(mqboolean disableCache);
```

Parameters

disableCache

mqboolean as TRUE or FALSE.

Returns

mqboolean value to indicate success or failure.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getAutoDispatch

Returns the value of the auto dispatch bool for this Connection

Declaration

```
mqcstring getAutoDispatch();
```

Returns

True or false as mqcstring.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getBatchTimeoutmqinterval

Gets the timeout interval in batching of message for performance mode

Declaration

```
mqcstring getBatchTimeoutmqinterval()
                        throw (CJMSException*);
```

Returns

Batch timeout interval in mqcstring .

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getCreateLocalSocket

Gets the value of the local socket bool.

Declaration

```
mqcstring getCreateLocalSocket()
                        throw (CJMSException*);
```

Returns

True or false as mqcstring.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getAdminConnectionReconnectmqinterval

Returns the value for the reconnect interval between two reconnect

attempts made all durable connections created using this Connection

Factory.

Declaration

```
mqlong getAdminConnectionReconnectmqinterval()
throw (CJMSException*);
```

Returns

Reconnect interval as mqlong.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getDurableConnectionReconnectmqinterval

Returns the value for the reconnect interval between two reconnect

attempts made all durable connections created using this Connection

Factory.

Declaration

```
mqlong getDurableConnectionReconnectmqinterval()
                        throw (CJMSException*);
```

Returns

Reconnect interval as mqlong.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getClientProxyURL

Returns Client Proxy URL being used for TCP based connections created using this ConnectionFactory.

Declaration

```
mqcstring getClientProxyURL();
```

Returns

URL of the client proxy as mqcstring.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getCompressionManager

Gets the CompressionManager class name to be used for MessageCompression purposes. The default value of the CompressionManager is fiorano.jms.services.msg.compression.def.CompressionManagerImpl Returns null if not set.

Declaration

```
mqcstring getCompressionManager();
```

Returns

Compression manager class name as mqcstring otherwise NULL.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getConnectionClientID

Returns the ClientID for this MetaData

Declaration

```
mqcstring getConnectionClientID();
```

Returns

Client ID as mqcstring.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getConnectURL

Returns the URL of the FioranoMQ server to which all connections created using this ConnectionFactory will be made.

Declaration

```
mqcstring getConnectURL();
```

Returns

Connect url as mqcstring.

getCSPUpdateFrequency

Gets the UpdateFrequency set for this connection factory. This value will be used by the CSP to pass this frequency to the local cache.

Declaration

```
mqcstring getCSPUpdateFrequency();
```

Returns

Update frequency value as mqcstring.

getDurableConnectionBaseDir

Gets the Durable Connection base directory for this ConnectionFactory as a mqcstring.

Declaration

```
mqcstring getDurableConnectionBaseDir()
throw (CJMSException *);
```

Returns

Path of the base directory that is being used for client side storage of messages as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getFactoryMetadataParams

Returns a Hashtable of all the properties for this CTopicConnectionFactory.

Declaration

```
CHashTable* getFactoryMetadataParams () throw (CJMSException *);
```

Returns

MetaData params as key-value pairs in CHashTable*.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getHTTPProxyURL

Returns HTTP Proxy URL being used for HTTP based connections created using this ConnectionFactory.

Declaration

```
mqcstring getHTTPProxyURL () throw (CJMSException *);
```

Returns

URL of the HTTP Proxy as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getLazyRSCreation

Returns the value of lazy creation of Receiver Socket

Declaration

```
mqboolean getLazyRSCreation () throw (CJMSException *);
```

Returns

true or false as mqboolean

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getLMSEnabled

Returns whether support for large messages is enabled or not

Declaration

```
mqboolean getLMSEnabled () throw (CJMSException *);
```

Returns

mqboolean true if large message support is enabled, false otherwise

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getLookUpPreferredServer

Gets the value for the lookup preferred server bool.

Declaration

```
mqcstring getLookUpPreferredServer () throw (CJMSException *);
```

Returns

true or false as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getMaxAdminConnectionReconnectAttempts

Returns the value for the Maximum number of reconnect attempts (for reconnecting to the FioranoMQ server)that are made by a admin connection created using this Connection Factory, as mqint.

Declaration

```
mqint getMaxAdminConnectionReconnectAttempts ()
                                    throw (CJMSException *);
```

Returns

maximum number of reconnect attempts as mqint

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getMaxDurableConnectionReconnectAttempts

Returns the value for the Maximum number of reconnect attempts (for reconnecting to the FioranoMQ server)that are made by a durable connection created using this Connection Factory, as mqint.

Declaration

```
mqint getMaxDurableConnectionReconnectAttempts()
throw (CJMSException *);
```

Returns

maximum number of reconnect attempts as mqint

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getMaxSocketCreationTries

Returns the value for the Maximum socket creation tries for all connections made using this ConnectionFactory.

Declaration

```
mqcstring getMaxSocketCreationTries()
throw (CJMSException *);
```

Returns

max socket creation tries as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getName

Returns the mqcstring name of the Meta data

Declaration

```
mqcstring getName()
throw (CJMSException *);
```

Returns

name of the metadata object as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getPort

Get the port of the Server on which the ConnectionFactory resides in the Server.

Declaration

```
mqint getPort () throw (CJMSException *);
```

Returns

Port number as mqint

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getProxyCredentials

Returns the password for the proxy through which HTTP based connections created using this ConnectionFactory are routed.

Declaration

```
mqcstring getProxyCredentials () throw (CJMSException *);
```

Returns

password of the proxy as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getProxyPrincipal

Returns the username for the proxy through which HTTP based connections created using this ConnectionFactory are routed.

Declaration

```
mqcstring getProxyPrincipal () throw (CJMSException *);
```

Returns

username of the proxy as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getProxyType

Returns the proxy name through which HTTP based connections created using this ConnectionFactory are routed.

Declaration

```
mqcstring getProxyType () throw (CJMSException *);
```

Returns

name of the proxy as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getPublishBehaviourInAutoRevalidation

Gets the behaviour of the publisher while in auto-revalidation.

Declaration

```
mqcstring getPublishBehaviourInAutoRevalidation () throw (CJMSException *);
```

Returns

publisher behaviour in mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getPublishWaitDuringCSPSyncp

Gets the delay in publishing new messages when CSP is syncing up.

Declaration

```
mqcstring getPublishWaitDuringCSPSyncp () throw (CJMSException *);
```

Returns

mqcstring - CSP syncup delay

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getSecondaryConnectURL

Get the semicolon seperated Secondary Connection URLs for this connection factory. If secondary URL doesnot exist, it returns null. This API is supported only for backward compatibility.

Declaration

```
mqcstring getSecondaryConnectURL () throw (CJMSException *);
```

Returns

Secondary connect url as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getSecurityProtocol

Returns the security protocol for this ConnectionFactory.

Declaration

```
mqcstring getSecurityProtocol () throw (CJMSException *);
```

Returns

Security protocol name as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getServerProxyURL

Returns Server Proxy URL being used for TCP based connections created using this ConnectionFactory.

Declaration

```
mqcstring getServerProxyURL () throw (CJMSException *);
```

Returns

URL of the Server Proxy as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getSleepSocketCreationTries

Returns the value for the sleep time between 2 socket creation tries for all connections] made using this ConnectionFactory.

Declaration

```
mqcstring getSleepSocketCreationTries () throw (CJMSException *);
```

Returns

Sleep time bewteen socket creation tries as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getSocketTimeout

Returns the timeout value for the sockets of all connections created using this ConnectionFactory.

Declaration

```
mqcstring getSocketTimeout () throw (CJMSException *);
```

Returns

timeout value as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getSOCKSProxyURL

Returns SOCKS Proxy URL being used for HTTP based connections created using this ConnectionFactory.

Declaration

```
mqcstring getSOCKSProxyURL () throw (CJMSException *);
```

Returns

URL of the SOCKS Proxy as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getStateTransitionOnReceiveSocket

checks if StateTransitionOnReceiveSocket is Enabled/Disabled

Declaration

```
mqboolean getStateTransitionOnReceiveSocket ()throw (CJMSException *);
```

Returns

true or false as mqboolean

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getTCPBatchSize

Gets the message batch size for this ConnectionFactory

Declaration

```
mqcstring getTCPBatchSize () throw (CJMSException *);
```

Returns

message batch size as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getTransportProtocol

Returns the Transport Protocol for this ConnectionFactory.

Declaration

```
mqcstring getTransportProtocol () throw (CJMSException *);
```

Returns

Transport Protocol as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

isAutoRevalidationEnabled

Returns whether auto-revalidation is enabled or not. Enabling auto-revalidation does not enable durable connections. Hence, local caching will not work if auto-revalidation is enabled.

Declaration

```
mqcstring isAutoRevalidationEnabled () throw (CJMSException *);
```

Returns

return mqcstring true if auto-revalidation is enabled, false otherwise

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

isBatchingEnabled

Check whether batching is enabled or not

Declaration

```
mqboolean isBatchingEnabled () throw (CJMSException *);
```

Returns

true or false as mqboolean

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

isConnectURLUpdationAllowed

Indicates whether Connect URL can be updated or not

Declaration

```
mqboolean isConnectURLUpdationAllowed () throw (CJMSException *);
```

Returns

true or false as mqboolean

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

isCSPStoredMessageSendDisabled

Checks if send of pending messages is disabled for this connection factory.

Declaration

```
mqcstring isCSPStoredMessageSendDisabled () throw (CJMSException *);
```

Returns

true if send pending is disabled else false. Null is returned if no value was set

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

isPingDisabled

Checks if ping is enabled or disabled for this ConnectionFactory.

Declaration

```
mqboolean isPingDisabled () throw (CJMSException *);
```

Returns

true if ping was disabled while creating the connection factory and false if otherwise

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

isReaderCacheDisabled

Tells whether the reader cache has been disabled for this connection factory. Disabling the reader cache causes the memory to be allocated for read operation.

Declaration

```
mqboolean isReaderCacheDisabled () throw (CJMSException *);
```

Returns

true or false as mqboolean

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

isSingleSocketForSendReceiveEnabled

checks if UseSingleSocketForSendReceive is Enabled/Disabled

Declaration

```
mqboolean isSingleSocketForSendReceiveEnabled () throw (CJMSException *);
```

Returns

true or false as mqboolean

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

isSocketKeepAliveEnabled

checks if SocketKeepAlive is Enabled/Disabled

Declaration

```
mqboolean isSocketKeepAliveEnabled () throw (CJMSException *);
```

Returns

true or false as mqboolean

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setAdminConnectionReconnectmqinterval

Sets the value for the reconnect interval which is the interval between

two successive reconnect attempts that would be made by a Admin

connection created using this Connection Factory. The value passed

should be positive in the range if a Long value.

Declaration

```
mqboolean setAdminConnectionReconnectmqinterval(mqcstring mqinterval)
                        throw (CJMSException*);
```

Parameters

mqinterval

The reconnect interval passed as mqcstring

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

setAutoDispatch

Sets the value for the auto dispatch bool. If set to true all connections created using this ConectionFactory ignore the dispatcher is enabled and get connected to the FioranoMQ server running on the ConnectURL of this ConnectionFactory.

Declaration

```
mqboolean setAutoDispatch(mqcstring autoDispatch);
```

Parameters

autoDispatch

autoDespatch in mqcstring having true or false value.

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setBatchTimeoutmqinterval

Sets the delay in publishing new messages when CSP is syncing up.

Declaration

```
mqboolean setBatchTimeoutmqinterval(mqcstring batchTimeoutmqinterval)
                        throw (CJMSException*);
```

Parameters

batchTimeoutmqinterval

the delay in publishing new messages when CSP is syncing up

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

setCreateLocalSocket

Sets the value for the local socket bool. If set to true this enables

the client server connection (if they are running on the same machine)

to be alive even if the machine is disconnected from the network.

Default value is false.

Declaration

```
mqboolean setCreateLocalSocket(mqcstring localSocket)
                        throw (CJMSException*);
```

Parameters

localSocket

True or false as mqcstring.

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

setDurableConnectionReconnectmqinterval

Sets the value for the reconnect interval which is the interval between

two successive reconnect attempts that would be made by a Durable

connection created using this Connection Factory. The value passed

should be positive in the range if a Long value.

Declaration

```
mqboolean setDurableConnectionReconnectmqinterval(mqcstring mqinterval)
throw (CJMSException*);
```

Parameters

mqinterval

The reconnect interval passed as mqcstring

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

setSecondaryConnectURL

Sets the back up connect url at position num for this Connection Factory.

Declaration

```
mqboolean setSecondaryConnectURL(mqcstring strSecondaryConnectURL);
```

Parameters

Back up connect URL in mqcstring

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setBatchingEnabled

Enable/Disable Batching

Declaration

```
mqboolean setBatchingEnabled(mqboolean enable);
```

Parameters

enable true or false to enable batching

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setClientProxyURL

Sets the URL for the Client side Proxy through which TCP based Connections created using this ConnectionFactory will be routed using HTTP Tunneling. This URL will be used only when the Transport Protocol is set as TCP.

Declaration

```
mqboolean setClientProxyURL(mqcstring proxyURL);
```

Parameters

proxyURL

URL of the Client Proxy as mqcstring

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setCompressionManager

Sets the CompressionManager class name to be used for MessageCompresson purposes. The default value of the CompressionManager is fiorano.jms.services.msg.compression.def.CompressionManagerImpl

Declaration

```
mqboolean setCompressionManager(mqcstring manager);
```

Parameters

manager

class name of the compression manager as mqcstring

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setConnectionClientID

Sets the ClientID for this MetaData. All connections created using this ConnectionFactory will have this ClientID.

Declaration

```
mqboolean setConnectionClientID(mqcstring clientID);
```

Parameters

clientID

client ID as mqcstring

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setConnectURL

Sets the primary Connect URL for this ConnectionFactory.

Declaration

```
mqboolean setConnectURL(mqcstring connectURL);
```

Parameters

connectURL

connection URL in mqcstring

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

setCSPUpdateFrequency

Sets the UpdateFrequency to the value passed. This will be used while creating a connection factory. All connections created using this connection factory will use this frequency for CSP if it is enabled on the connection

Declaration

```
mqboolean setCSPUpdateFrequency(mqcstring frequency);
```

Parameters

frequency

the CSP Update frequency as mqcstring

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setDurableConnectionBaseDir

Sets the Durable Connection Base directory for this ConnectionFactory as a mqcstring. This directory will be used for the client side caching of messages for all connections created using this ConnectionFactory.

Declaration

```
mqboolean setDurableConnectionBaseDir(mqcstring durBaseDir);
```

Parameters

durBaseDir

Fully qualified path of the base directory for client side storage of messages

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setFactoryMetadataParams

Sets the Hashtable containing all properties for this Connection Factory. This API can be used to set all required properties in one call in the ConnectionFactory rather than making each set call individually.

Declaration

```
mqboolean setFactoryMetadataParams(CHashTable *Params);
```

Parameters

Params

hashtable containing the name value pairs of all Connection Factory properties.

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setHTTPProxyURL

Sets the URL for the HTTP Proxy through which HTTP based Connections created using this ConnectionFactory will be routed. This URL will be used only when the Transport Protocol is set as HTTP.

Declaration

```
mqboolean setHTTPProxyURL(mqcstring proxyURL);
```

Parameters

proxyURL

proxy URL as mqcstring

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setIsForLPC

sets Is for LPC for object

Declaration

```
mqboolean setIsForLPC(mqboolean isForLPC);
```

Parameters

isForLPC

true or false as mqboolean

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setLazyRSCreation

Enables/disables Lazy ReceiveWorker creation

Declaration

```
mqboolean setLazyRSCreation(mqboolean lazyRS);
```

Parameters

lazyRS

lazyRS mqboolean value

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setLMSEnabled

Enables/disables large message support

Declaration

```
mqboolean setLMSEnabled(mqboolean isLMSEnabled);
```

Parameters

isLMSEnabled

bool true if Large support has to be enabled, false otherwise

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setLookUpPreferredServer

Sets the value for the lookup preferred server bool. If this is set to true all connections made using this ConnectionFactory will be routed to the preferred server in a dispatcher cluster.

Declaration

```
mqboolean setLookUpPreferredServer(mqcstring lookupPreferred);
```

Parameters

lookupPreferred

true or false passed as mqcstring

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setMaxAdminConnectionReconnectAttempts

Sets the value for the Maximum number of reconnect attempts (for reconnecting to the FioranoMQ server)that will be made by an admin connection created using this Connection Factory. The value passed should be positive in the range of an Integer value.

Declaration

```
mqboolean setMaxAdminConnectionReconnectAttempts(mqcstring numAttempts);
```

Parameters

numAttempts

maximum number of reconnect attempts passed as mqcstring

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setMaxDurableConnectionReconnectAttempts

Sets the value for the Maximum number of reconnect attempts (for reconnecting to the FioranoMQ server) that will be made by a durable connection created using this Connection Factory. The value passed should be positive in the range of an Integer value.

Declaration

```
mqboolean setMaxDurableConnectionReconnectAttempts(mqcstring numAttempts);
```

Parameters

numAttempts

maximum number of reconnect attempts passed as mqcstring

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setMaxSocketCreationTries

Sets the value for the maximum socket creation tries that the FioranoMQ runtime will make for creating the socket for all connections made using this ConnectionFactory. The value should be in the range of an int value

Declaration

```
mqboolean setMaxSocketCreationTries(mqcstring socketTries);
```

Parameters

socketTries

socketTries passed as mqcstring

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setName

Set the Name of this MetaData Object (uppercase)

Declaration

```
mqboolean setName(mqcstring metaDataName)
throw (CJMSException *);
```

Parameters

metaDataName

Name as mqcstring

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setProxyCredentials

Sets the password for getting validated at the proxy (if required) which is to be used for the HTTP based connections created using this ConnectionFactory. This property is used only when the Transport Protocol is set as HTTP.

Declaration

```
mqboolean setProxyCredentials(mqcstring password);
```

Parameters

password

Password as mqcstring

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setProxyPrincipal

Sets the username for the proxy (if required) through which HTTP based connections created using this ConnectionFactory are routed. This property is used only when the Transport Protocol is set as HTTP.

Declaration

```
mqboolean setProxyPrincipal(mqcstring username);
```

Parameters

username

Username as mqcstring

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setProxyType

Sets the type (name) of the proxy being used for the HTTP based connections created using this ConnectionFactory. This property is used only when the Transport Protocol is set as HTTP. The possible values for the ProxyType can be accessed from the MetaDataConstants interface.

Declaration

```
mqboolean setProxyType(mqcstring proxy);
```

Parameters

proxy

Proxy type as mqcstring

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setPublishBehaviourInAutoRevalidation

Sets the delay in publishing new messages when CSP is syncing up.

Declaration

```
mqboolean setPublishBehaviourInAutoRevalidation(mqcstring publishBehaviour);
```

Parameters

publishBehaviour

publisher behaviour on auto-revalidation as mqcstring

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setPublishWaitDuringCSPSyncp

Sets the delay in publishing new messages when CSP is syncing up.

Declaration

```
mqboolean setPublishWaitDuringCSPSyncp(mqcstring delay);
```

Parameters

delay

delay time in mqcstring

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setSecondaryConnectURL

Set the secondary Connection URL for this Connection Factory. If no secondary Connection URL exists for this connection factory, it creates new Connection Factory URLs list from the semi colon seperated param. Otherwise it replaces the existing secondaryConnectURL by the params. This API is supported only for backward compatibility. Please Use setBackupConnectURLs(mqcstring param)for future use.

Declaration

```
mqboolean setSecondaryConnectURL(mqcstring strSecondaryConnectURL);
```

Parameters

strSecondaryConnectURL

strSecondaryConnectURL The new secondaryConnectURL value

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setSecurityProtocol

Sets the security protocol over which all connections created using this ConnectionFactory communicate with the FioranoMQ server

Declaration

```
mqboolean setSecurityProtocol(mqcstring protocol);
```

Parameters

protocol

Security Protocol as mqcstring

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setServerProxyURL

Sets the URL for the Server side Proxy through which TCP based Connections created using this ConnectionFactory will be routed using HTTP Tunneling. This URL will be used only when the Transport Protocol is set as TCP.

Declaration

```
mqboolean setServerProxyURL(mqcstring proxyURL);
```

Parameters

proxyURL

Proxy URL as mqcstring

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setSleepSocketCreationTries

Sets the value for the sleep time bewteen 2 creation tries that the FioranoMQ runtime will make for creating the socket for all connections made using this ConnectionFactory. The value should be in the range of a long value

Declaration

```
mqboolean setSleepSocketCreationTries(mqcstring sleepTime);
```

Parameters

sleepTime

Sleep time as mqcstring

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setSocketKeepAlive

Enable/Disable Socket SetKeepAlive

Declaration

```
mqboolean setSocketKeepAlive(mqboolean enable);
```

Parameters

enable

true or false as mqcstring

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setSocketTimeout

Sets the timeout value for the sockets of all connections created using this ConnectionFactory. The value should not be negative and should be the range of a long value.

Declaration

```
mqboolean setSocketTimeout(mqcstring timeout);
```

Parameters

timeout

timeout value as mqcstring

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setSOCKSProxyURL

Sets the URL for the SOCKS Proxy through which HTTP based Connections created using this ConnectionFactory will be routed. This URL will be used only when the Transport Protocol is set as HTTP.

Declaration

```
mqboolean setSOCKSProxyURL(mqcstring proxyURL);
```

Parameters

proxyURL

proxyURL as mqcstring

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setStateTransitionOnReceiveSocket

Enable/Disable StateTransitionOnReceiveSocket

Declaration

```
mqboolean setStateTransitionOnReceiveSocket(mqboolean enable);
```

Parameters

enable

true or false as mqboolean

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setTCPBatchSize

Sets the value of the message batch size that will be used for NP messages being sent using all connections created using this Connection Factory. This is done to utilize the TCP/IP buffer size to the maximum.

Declaration

```
mqboolean setTCPBatchSize(mqcstring batchSize);
```

Parameters

batchSize

the message batch size that will be used for

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setTransportProtocol

Sets the transport protocol over which all connections created using this ConnectionFactory communicate with the FioranoMQ server.

Declaration

```
mqboolean setTransportProtocol(mqcstring protocol);
```

Parameters

protocol

Transport protocol as mqcstring

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setUseSingleSocketForSendReceive

Enable/Disable UseSingleSocketForSendReceive

Declaration

```
mqboolean setUseSingleSocketForSendReceive(mqboolean enable);
```

Parameters

enable

true or false as mqboolean

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

updateConnectURL

Sets a bool in ConnectionFactory indicating whether Connect URL can be updated or not

Declaration

```
mqboolean updateConnectURL(mqboolean flag);
```

Parameters

flag

true or false value as mqboolean

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

CQueueConnectionFactoryMetaData

Class CQueueConnectionFactoryMetaData represents the MetaData information for a CQueueConnectionFactory at it is stored in an LDAP directory. Objects of this class are used to create CQueueConnectionFactory objects.

allowAutoRevalidation

Sets the value for auto-revalidation. Setting this to true revalidates all the connections automatically whenever connection with the server goes down. This method does not allow one to set durable connections to true, i.e., no local caching of messages is possible using this API

Declaration

```
mqboolean allowAutoRevalidation(mqcstring allowAutoRevaidation);
```

Parameters

allowAutoRevaidation

mqboolean to indicate allowAutoRevaidation setting as TRUE or FALSE.

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

allowDurableConnections

Enables the durable connections for this ConnectionFactory. This means that all connections created using this ConnectionFactory will be durable connections if this support is enabled in the FioranoMQ Server.

Declaration

```
mqboolean allowDurableConnections(mqboolean allowDurableConnections);
```

Parameters

allowDurableConnections

mqboolean to indicate allowDurableConnections setting as TRUE or FALSE.

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

areDurableConnectionsAllowed

Gives the status whether durable connection are allowed or not for this connection factory.

Declaration

```
mqboolean areDurableConnectionsAllowed () throw (CJMSException *);
```

Returns

mqboolean value for the status whether durable connection are allowed or not for this connection factory.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

compareConnectURLs

Compares the ConnectURLS in the CQueueConnectionFactoryMetaData.

Declaration

```
mqboolean compareConnectURLs(CQueueConnectionFactoryMetaData* metaData);
```

Parameters

metadata

CQueueConnectionFactoryMetaData pointer.

Returns

mqboolean value to indicate if the urls are equal or not equal.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

disableCSPStoredMessageSend

Disables the sending of any pending messages in the client side store of all durable connections created using this ConnectionFactory. If no value is set then send of pending messages is enabled.

Declaration

```
mqboolean disableCSPStoredMessageSend(mqcstring dontSend);
```

Parameters

dontSend

mqcstring as TRUE or FALSE.

Returns

mqboolean value to indicate if the urls are equal or not equal.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

disablePing

Disables the ping functionality for this CQueueConnectionFactory. This would mean that no pinging would be done by the FioranoMQ runtime for all connections created using this CQueueConnectionFactory even if pinging is enabled in the FioranoMQ server.

Declaration

```
mqboolean disablePing(mqboolean ping);
```

Parameters

Ping

mqboolean as TRUE or FALSE.

Returns

mqboolean value to indicate success or failure.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

disableReaderCache

Disables the reader cache in the connection buffers. The memory will be re-allocated for each new read sequence on each of the connection created out of this connection factory.

Declaration

```
mqboolean disableReaderCache(mqboolean disableCache);
```

Parameters

disableCache

mqboolean as TRUE or FALSE.

Returns

mqboolean value to indicate success or failure.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getAdminConnectionReconnectmqinterval

Returns the value for the reconnect interval between two reconnect

attempts made all durable connections created using this Connection

Factory.

Declaration

```
mqlong getAdminConnectionReconnectmqinterval()
```

throw (CJMSException*);

Returns

Reconnect interval as mqlong.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getAutoDispatch

Returns the value of the auto dispatch bool for this Connection

Declaration

```
mqcstring getAutoDispatch () throw (CJMSException *);
```

Returns

True or false as mqcstring.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getBatchTimeoutmqinterval

Gets the timeout interval in batching of message for performance mode.

Declaration

```
mqcstring getBatchTimeoutmqinterval()
                        throw (CJMSException*);
```

Returns

Batch timeout interval in string .

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getClientProxyURL

Returns Client Proxy URL being used for TCP based connections created using this ConnectionFactory.

Declaration

```
mqcstring getClientProxyURL () throw (CJMSException *);
```

Returns

URL of the client proxy as mqcstring.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getCompressionManager

Gets the CompressionManager class name to be used for MessageCompression purposes. The default value of the CompressionManager is fiorano.jms.services.msg.compression.def.CompressionManagerImpl Returns null if not set.

Declaration

```
mqcstring getCompressionManager () throw (CJMSException *);
```

Returns

Compression manager class name as mqcstring otherwise NULL.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getConnectionClientID

Returns the ClientID for this MetaData

Declaration

```
mqcstring getConnectionClientID () throw (CJMSException *);
```

Returns

Client ID as mqcstring.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getConnectURL

Returns the URL of the FioranoMQ server to which all connections created using this ConnectionFactory will be made.

Declaration

```
mqcstring getConnectURL () throw (CJMSException *);
```

Returns

Connect url as mqcstring.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getCreateLocalSocket

Gets the value of the local socket bool.

Declaration

```
mqcstring getCreateLocalSocket () throw (CJMSException *);
```

Returns

True or false as mqcstring.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getCSPUpdateFrequency

Gets the UpdateFrequency set for this connection factory. This value will be used by the CSP to pass this frequency to the local cache.

Declaration

```
mqcstring getCSPUpdateFrequency () throw (CJMSException *);
```

Returns

Update frequency value as mqcstring.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getDurableConnectionBaseDir

Gets the Durable Connection base directory for this ConnectionFactory as a mqcstring.

Declaration

```
mqcstring getDurableConnectionBaseDir () throw (CJMSException *);
```

Returns

Path of the base directory that is being used for client side storage of messages as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getDurableConnectionReconnectmqinterval

Returns the value for the reconnect interval between two reconnect

attempts made all durable connections created using this Connection

Factory.

Declaration

```
mqlong getDurableConnectionReconnectmqinterval()
        throw (CJMSException*);
```

Returns

Reconnect interval as mqlong.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getFactoryMetadataParams

Returns a Hashtable of all the properties for this CQueueConnectionFactory.

Declaration

```
CHashTable* getFactoryMetadataParams () throw (CJMSException *);
```

Returns

MetaData params as key-value pairs in CHashTable*.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getHTTPProxyURL

Returns HTTP Proxy URL being used for HTTP based connections created using this ConnectionFactory.

Declaration

```
mqcstring getHTTPProxyURL () throw (CJMSException *);
```

Returns

URL of the HTTP Proxy as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getLazyRSCreation

Returns the value of lazy creation of Receiver Socket

Declaration

```
mqboolean getLazyRSCreation () throw (CJMSException *);
```

Returns

true or false as mqboolean

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getLMSEnabled

Returns whether support for large messages is enabled or not

Declaration

```
mqboolean getLMSEnabled () throw (CJMSException *);
```

Returns

mqboolean true if large message support is enabled, false otherwise

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getLookUpPreferredServer

Gets the value for the lookup preferred server bool.

Declaration

```
mqcstring getLookUpPreferredServer () throw (CJMSException *);
```

Returns

true or false as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getMaxAdminConnectionReconnectAttempts

Returns the value for the Maximum number of reconnect attempts (for reconnecting to the FioranoMQ server)that are made by a admin connection created using this Connection Factory, as mqint.

Declaration

```
mqint getMaxAdminConnectionReconnectAttempts () throw (CJMSException *);
```

Returns

maximum number of reconnect attempts as mqint

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getMaxDurableConnectionReconnectAttempts

Returns the value for the Maximum number of reconnect attempts (for reconnecting to the FioranoMQ server) that are made by a durable connection created using this Connection Factory, as mqint.

Declaration

```
mqint getMaxDurableConnectionReconnectAttempts () throw (CJMSException *);
```

Returns

maximum number of reconnect attempts as mqint

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getMaxSocketCreationTries

Returns the value for the Maximum socket creation tries for all connections made using this ConnectionFactory.

Declaration

```
mqcstring getMaxSocketCreationTries () throw (CJMSException *);
```

Returns

max socket creation tries as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getName

Returns the mqcstring name of the Meta data

Declaration

```
mqcstring getName()
throw (CJMSException *);
```

Returns

name of the metadata object as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getPort

Get the port of the Server on which the ConnectionFactory resides in the Server.

Declaration

```
mqint getPort () throw (CJMSException *);
```

Returns

Port number as mqint

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

getProxyCredentials

Returns the password for the proxy through which HTTP based connections created using this ConnectionFactory are routed.

Declaration

```
mqcstring getProxyCredentials () throw (CJMSException *);
```

Returns

password of the proxy as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getProxyPrincipal

Returns the username for the proxy through which HTTP based connections created using this ConnectionFactory are routed.

Declaration

```
mqcstring getProxyPrincipal () throw (CJMSException *);
```

Returns

username of the proxy as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getProxyType

Returns the proxy name through which HTTP based connections created using this ConnectionFactory are routed.

Declaration

```
mqcstring getProxyType () throw (CJMSException *);
```

Returns

name of the proxy as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getPublishBehaviourInAutoRevalidation

Gets the behaviour of the publisher while in auto-revalidation.

Declaration

```
mqcstring getPublishBehaviourInAutoRevalidation () throw (CJMSException *);
```

Returns

publisher behaviour in mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getPublishWaitDuringCSPSyncp

Gets the delay in publishing new messages when CSP is syncing up.

Declaration

```
mqcstring getPublishWaitDuringCSPSyncp () throw (CJMSException *);
```

Returns

mqcstring - CSP syncup delay

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getSecondaryConnectURL

Get the semicolon seperated Secondary Connection URLs for this connection factory. If secondary URL doesnot exist, it returns null. This API is supported only for backward compatibility.

Declaration

```
mqcstring getSecondaryConnectURL () throw (CJMSException *);
```

Returns

Secondary connect url as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getSecurityProtocol

Returns the security protocol for this ConnectionFactory.

Declaration

```
mqcstring getSecurityProtocol () throw (CJMSException *);
```

Returns

Security protocol name as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getServerProxyURL

Returns Server Proxy URL being used for TCP based connections created using this ConnectionFactory.

Declaration

```
mqcstring getServerProxyURL () throw (CJMSException *);
```

Returns

URL of the Server Proxy as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getSleepSocketCreationTries

Returns the value for the sleep time between 2 socket creation tries for all connections] made using this ConnectionFactory.

Declaration

```
mqcstring getSleepSocketCreationTries () throw (CJMSException *);
```

Returns

Sleep time bewteen socket creation tries as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getSocketTimeout

Returns the timeout value for the sockets of all connections created using this ConnectionFactory.

Declaration

```
mqcstring getSocketTimeout () throw (CJMSException *);
```

Returns

timeout value as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getSOCKSProxyURL

Returns SOCKS Proxy URL being used for HTTP based connections created using this ConnectionFactory.

Declaration

```
mqcstring getSOCKSProxyURL () throw (CJMSException *);
```

Returns

URL of the SOCKS Proxy as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getStateTransitionOnReceiveSocket

checks if StateTransitionOnReceiveSocket is Enabled/Disabled

Declaration

```
mqboolean getStateTransitionOnReceiveSocket () throw (CJMSException *);
```

Returns

true or false as mqboolean

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getTCPBatchSize

Gets the message batch size for this ConnectionFactory

Declaration

```
mqcstring getTCPBatchSize () throw (CJMSException *);
```

Returns

message batch size as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

getTransportProtocol

Returns the Transport Protocol for this ConnectionFactory.

Declaration

```
mqcstring getTransportProtocol () throw (CJMSException *);
```

Returns

Transport Protocol as mqcstring

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

isAutoRevalidationEnabled

Returns whether auto-revalidation is enabled or not. Enabling auto-revalidation does not enable durable connections. Hence, local caching will not work if auto-revalidation is enabled.

Declaration

```
mqcstring isAutoRevalidationEnabled () throw (CJMSException *);
```

Returns

return mqcstring true if auto-revalidation is enabled, false otherwise

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

isBatchingEnabled

Check whether batching is enabled or not

Declaration

```
mqboolean isBatchingEnabled () throw (CJMSException *);
```

Returns

true or false as mqboolean

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

isConnectURLUpdationAllowed

Indicates whether Connect URL can be updated or not

Declaration

```
mqboolean isConnectURLUpdationAllowed () throw (CJMSException *);
```

Returns

true or false as mqboolean

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

isCSPStoredMessageSendDisabled

Checks if send of pending messages is disabled for this connection factory.

Declaration

```
mqcstring isCSPStoredMessageSendDisabled () throw (CJMSException *);
```

Returns

true if send pending is disabled else false. Null is returned if no value was set

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

isForLPC

Gets the LPC attribute of the ConnectionFactoryMetaData object

Declaration

```
mqboolean isForLPC()
throw (CJMSException*);
```

Returns

True or false as string.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

isReaderCacheDisabled

Tells whether the reader cache has been disabled for this connection factory. Disabling the reader cache causes the memory to be allocated for read operation.

Declaration

```
mqboolean isReaderCacheDisabled () throw (CJMSException *);
```

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

isSingleSocketForSendReceiveEnabled

checks if UseSingleSocketForSendReceive is Enabled/Disabled

Declaration

```
mqboolean isSingleSocketForSendReceiveEnabled () throw (CJMSException *);
```

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

isSocketKeepAliveEnabled

checks if SocketKeepAlive is Enabled/Disabled

Declaration

```
mqboolean isSocketKeepAliveEnabled () throw (CJMSException *);
```

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setAdminConnectionReconnectmqinterval

Sets the value for the reconnect interval which is the interval between

two successive reconnect attempts that would be made by a Admin

connection created using this Connection Factory. The value passed

should be positive in the range if a Long value.

Declaration

```
mqboolean setAdminConnectionReconnectmqinterval(mqcstring mqinterval)
                        throw (CJMSException*);
```

Parameters

mqinterval

Reconnect interval passed as mqcstring.

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

setAutoDispatch

Sets the value for the auto dispatch bool. If set to true all connections created using this ConectionFactory ignore the dispatcher is enabled and get connected to the FioranoMQ server running on the ConnectURL of this ConnectionFactory.

Declaration

```
mqboolean setAutoDispatch(mqcstring autoDispatch);
```

Parameters

autoDispatch

autoDespatch in mqcstring having true or false value.

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setBatchingEnabled

Enable/Disable Batching

Declaration

```
mqboolean setBatchingEnabled(mqboolean enable);
```

Parameters

enable

true or false to enable batching

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setClientProxyURL

Sets the URL for the Client side Proxy through which TCP based Connections created using this ConnectionFactory will be routed using HTTP Tunneling. This URL will be used only when the Transport Protocol is set as TCP.

Declaration

```
mqboolean setClientProxyURL(mqcstring proxyURL);
```

Parameters

proxyURL

URL of the Client Proxy as mqcstring

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setCompressionManager

Sets the CompressionManager class name to be used for MessageCompresson purposes. The default value of the CompressionManager is fiorano.jms.services.msg.compression.def.CompressionManagerImpl

Declaration

```
mqboolean setCompressionManager(mqcstring manager);
```

Parameters

manager

class name of the compression manager as mqcstring

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

setConnectionClientID

Sets the ClientID for this MetaData. All connections created using this ConnectionFactory will have this ClientID.

Declaration

```
mqboolean setConnectionClientID(mqcstring clientID);
```

Parameters

clientID

client ID as mqcstring

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setConnectURL

Sets the primary Connect URL for this ConnectionFactory.

Declaration

```
mqboolean setConnectURL(mqcstring metaDataDescription);
```

Parameters

connectURL

connection URL in mqcstring

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setCreateLocalSocket

Sets the value for the local socket bool. If set to true this enables the client server connection (if they are running on the same machine) to be alive even if the machine is disconnected from the network. Default value is false.

Declaration

```
mqboolean setCreateLocalSocket(mqcstring localSocket);
```

Parameters

localSocket

localSocket in mqcstring

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setCSPUpdateFrequency

Sets the UpdateFrequency to the value passed. This will be used while creating a connection factory. All connections created using this connection factory will use this frequency for CSP if it is enabled on the connection

Declaration

```
mqboolean setCSPUpdateFrequency(mqcstring frequency);
```

Parameters

frequency

the CSP Update frequency as mqcstring

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setDurableConnectionBaseDir

Sets the Durable Connection Base directory for this ConnectionFactory as a mqcstring. This directory will be used for the client side caching of messages for all connections created using this ConnectionFactory.

Declaration

```
mqboolean setDurableConnectionBaseDir(mqcstring durBaseDir);
```

Parameters

durBaseDir

Fully qualified path of the base directory for client side storage of messages

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setDurableConnectionReconnectmqinterval

Sets the value for the reconnect interval which is the interval between

two successive reconnect attempts that would be made by a Durable

connection created using this Connection Factory. The value passed

should be positive in the range if a Long value

Declaration

```
mqboolean setDurableConnectionReconnectmqinterval(mqcstring mqinterval)
                        throw (CJMSException*);
```

Parameters

mqinterval

The reconnect interval passed as mqcstring.

Returns

mqboolean value for success or failure from the server

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

setFactoryMetadataParams

Sets the Hashtable containing all properties for this Connection Factory. This API can be used to set all required properties in one call in the ConnectionFactory rather than making each set call individually.

Declaration

```
mqboolean setFactoryMetadataParams(CHashTable *Params);
```

Parameters

Params

hashtable containing the name value pairs of all Connection Factory properties.

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setHTTPProxyURL

Sets the URL for the HTTP Proxy through which HTTP based Connections created using this ConnectionFactory will be routed. This URL will be used only when the Transport Protocol is set as HTTP.

Declaration

```
mqboolean setHTTPProxyURL(mqcstring proxyURL);
```

Parameters

proxyURL

proxy URL as mqcstring

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setIsForLPC

sets Is for LPC for object

Declaration

```
mqboolean setIsForLPC(mqboolean isForLPC);
```

Parameters

isForLPC

true or false as mqboolean

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setLazyRSCreation

Enables/disables Lazy ReceiveWorker creation

Declaration

```
mqboolean setLazyRSCreation(mqboolean lazyRS);
```

Parameters

lazyRS

lazyRS mqboolean value

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setLMSEnabled

Enables/disables large message support

Declaration

```
mqboolean setLMSEnabled(mqboolean isLMSEnabled);
```

Parameters

isLMSEnabled

bool true if Large support has to be enabled, false otherwise

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setLookUpPreferredServer

Sets the value for the lookup preferred server bool. If this is set to true all connections made using this ConnectionFactory will be routed to the preferred server in a dispatcher cluster.

Declaration

```
mqboolean setLookUpPreferredServer(mqcstring lookupPreferred);
```

Parameters

lookupPreferred

true or false passed as mqcstring

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setMaxAdminConnectionReconnectAttempts

Sets the value for the Maximum number of reconnect attempts (for reconnecting to the FioranoMQ server)that will be made by an admin connection created using this Connection Factory. The value passed should be positive in the range of an Integer value.

Declaration

```
mqboolean setMaxAdminConnectionReconnectAttempts(mqcstring numAttempts);
```

Parameters

numAttempts

maximum number of reconnect attempts passed as mqcstring

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setMaxDurableConnectionReconnectAttempts

Sets the value for the Maximum number of reconnect attempts (for reconnecting to the FioranoMQ server)that will be made by a durable connection created using this Connection Factory. The value passed should be positive in the range of an Integer value.

Declaration

```
mqboolean setMaxDurableConnectionReconnectAttempts(mqcstring numAttempts);
```

Parameters

numAttempts

maximum number of reconnect attempts passed as mqcstring

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setMaxSocketCreationTries

Sets the value for the maximum socket creation tries that the FioranoMQ runtime will make for creating the socket for all connections made using this ConnectionFactory. The value should be in the range of an int value

Declaration

```
mqboolean setMaxSocketCreationTries(mqcstring socketTries);
```

Parameters

socketTries

socketTries passed as mqcstring

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setName

Set the Name of this MetaData Object (uppercase)

Declaration

```
mqboolean setName(mqcstring metaDataName)

throw(CJMSException *);
```

Parameters

metaDataName

Name as mqcstring

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setProxyCredentials

Sets the password for getting validated at the proxy (if required) which is to be used for the HTTP based connections created using this ConnectionFactory. This property is used only when the Transport Protocol is set as HTTP.

Declaration

```
mqboolean setProxyCredentials(mqcstring password);
```

Parameters

password

Password as mqcstring

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setProxyPrincipal

Sets the username for the proxy (if required) through which HTTP based connections created using this ConnectionFactory are routed. This property is used only when the Transport Protocol is set as HTTP.

Declaration

```
mqboolean setProxyPrincipal(mqcstring username);
```

Parameters

username

Username as mqcstring

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setProxyType

Sets the type (name) of the proxy being used for the HTTP based connections created using this ConnectionFactory. This property is used only when the Transport Protocol is set as HTTP. The possible values for the ProxyType can be accessed from the MetaDataConstants interface.

Declaration

```
mqboolean setProxyType(mqcstring proxy);
```

Parameters

proxy

Proxy type as mqcstring

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setPublishBehaviourInAutoRevalidation

Sets the delay in publishing new messages when CSP is syncing up.

Declaration

```
mqboolean setPublishBehaviourInAutoRevalidation(mqcstring publishBehaviour);
```

Parameters

publishBehaviour

publisher behaviour on auto-revalidation as mqcstring

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setPublishWaitDuringCSPSyncp

Sets the delay in publishing new messages when CSP is syncing up.

Declaration

```
mqboolean setPublishWaitDuringCSPSyncp(mqcstring delay);
```

Parameters

delay

delay time in mqcstring

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setSecondaryConnectURL

Set the secondary Connection URL for this Connection Factory. If no secondary Connection URL exists for this connection factory, it creates new Connection Factory URLs list from the semi colon seperated param. Otherwise it replaces the existing secondaryConnectURL by the params. This API is supported only for backward compatibility. Please Use setBackupConnectURLs (mqcstring param) for future use.

Declaration

```
mqboolean setSecondaryConnectURL(mqcstring strSecondaryConnectURL);
```

Parameters

strSecondaryConnectURL

strSecondaryConnectURL The new secondaryConnectURL value

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setSecurityProtocol

Sets the security protocol over which all connections created using this ConnectionFactory communicate with the FioranoMQ server

Declaration

```
mqboolean setSecurityProtocol(mqcstring protocol);
```

Parameters

protocol

Security Protocol as mqcstring

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setServerProxyURL

Sets the URL for the Server side Proxy through which TCP based Connections created using this ConnectionFactory will be routed using HTTP Tunneling. This URL will be used only when the Transport Protocol is set as TCP.

Declaration

```
mqboolean setServerProxyURL(mqcstring proxyURL);
```

Parameters

proxyURL

Proxy URL as mqcstring

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setSleepSocketCreationTries

Sets the value for the sleep time bewteen 2 creation tries that the FioranoMQ runtime will make for creating the socket for all connections made using this ConnectionFactory. The value should be in the range of a long value

Declaration

```
mqboolean setSleepSocketCreationTries(mqcstring sleepTime);
```

Parameters

sleepTime

Sleep time as mqcstring

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setSocketKeepAlive

Enable/Disable Socket SetKeepAlive

Declaration

```
mqboolean setSocketKeepAlive(mqboolean enable);
```

Parameters

enable

true or false as mqcstring

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setSocketTimeout

Sets the timeout value for the sockets of all connections created using this ConnectionFactory. The value should not be negative and should be the range of a long value.

Declaration

```
mqboolean setSocketTimeout(mqcstring timeout);
```

Parameters

timeout

timeout value as mqcstring

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setSOCKSProxyURL

Sets the URL for the SOCKS Proxy through which HTTP based Connections created using this ConnectionFactory will be routed. This URL will be used only when the Transport Protocol is set as HTTP.

Declaration

```
mqboolean setSOCKSProxyURL(mqcstring proxyURL);
```

Parameters

proxyURL

proxyURL as mqcstring

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setStateTransitionOnReceiveSocket

Enable/Disable StateTransitionOnReceiveSocket

Declaration

```
mqboolean setStateTransitionOnReceiveSocket(mqboolean enable);
```

Parameters

enable

true or false as mqboolean

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setTCPBatchSize

Sets the value of the message batch size that will be used for NP messages being sent using all connections created using this Connection Factory. This is done to utilize the TCP/IP buffer size to the maximum.

Declaration

```
mqboolean setTCPBatchSize(mqcstring batchSize);
```

Parameters

batchSize

the message batch size that will be used for

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setTransportProtocol

Sets the transport protocol over which all connections created using this ConnectionFactory communicate with the FioranoMQ server.

Declaration

```
mqboolean setTransportProtocol(mqcstring protocol);
```

Parameters

protocol

Transport protocol as mqcstring

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

setUseSingleSocketForSendReceive

Enable/Disable UseSingleSocketForSendReceive

Declaration

```
mqboolean setUseSingleSocketForSendReceive(mqboolean enable);
```

Parameters

enable

true or false as mqboolean

Returns

mqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some

internal error.

updateConnectURL

Sets a bool in ConnectionFactory indicating whether Connect URL can be updated or not

Declaration

```
mqboolean updateConnectURL(mqboolean flag);
```

Parameters

flag

true or false value as mqboolean

Returns

mqbooleanmqboolean value for success or failure from the server.

Throws

CJMSException

if the JMS provider fails to close the connection due to some internal error.

Chapter 6: Using Sample Programs

This chapter explains the various steps involved in running the sample programs which are shipped as part of the installer.

Organization of Samples Provided

The samples programs illustrating the use of C++RTL for PubSub and PTP Operations are organized into two categories.

- **Pubsub:** This directory contains the following sample programs, which illustrate basic JMS Publish/Subscribe functionality, using the C++RTL.
- **Ptp:** This directory contains two sample programs which illustrate JMS Request-Reply mechanism using the C++RTL.

Compiling and Running the Samples

To run the samples using FioranoMQ, perform the following steps:

Compile each of the source files, using the script file, cppclientbuild.bat, in the fmq/clients/cpp/native/scripts folder. Environment settings like path, compiler settings can be modified in the cppclientbuild.bat file.

For information on compiling and running these samples please refer to the readme file in the cpp\native\samples directory of FioranoMQ installation.

Limitations of C++RTL

Exception handling is limited to CJMSException and any variation is implemented by varying the error Code and error description parameters.

C++RTL provides a limited set of APIs for the end user, which just allows messaging using Pub/Sub or Sender/Receiver.

Chapter 7: Native C++ Runtime Examples

This chapter describes usage of C++ Runtime for connecting to FioranoMQ server. The various sample programs illustrates the use of simple Publish-Subscribe and Point-To-Point operations.

PTP

This directory contains samples which demonstrate the following functionality over PTP using C++ Runtime.

- Admin
- Basic Send Receive
- Browser
- HTTP
- HTTPS
- Message Selector
- Multi-thread PTP
- RequestReply
    - Basic
    - TimedRequestReply
- SSL
- Transaction

PubSub

This directory contains samples which demonstrate the following functionality over PubSub using C++ Runtime.

- Admin
- Basic Pub Sub
- Durable Subscriber
- HTTP
- HTTPS
- Message Selector
- Multi-thread PubSub
- RequestReply
    - Basic
    - TimedRequestReply
- SSL
- Transaction

These samples are available in %FMQ_DIR%\clients\cpp\native\samples directory. The %FMQ_DIR%\clients\cpp\native\script directory contains a script called build_samples.bat which compiles the C++ programs.

PTP Samples

Admin

This directory contains one sample program which illustrates basic JMS Administration API functionality using the FioranoMQ C++ Runtime Library.

- **AdminTest.cpp** - Creates an Admin Connection with the MQServer and gets an MQAdminService object to create and delete Queues and QueueConnectionFacotries and retrieves information of users connected from the server.

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. The %FMQ_DIR%\clients\cpp\native\scripts directory contains a script called cppclientbuild.bat which compiles the C++ program.
2. Run the AdminTest by executing the *AdminTest.exe* executable file.

Note: To run any of the C++ samples, please ensure that environment variable FMQ_DIR points to Fiorano's installation directory. (**Default:** C:\PROGRA~1\Fiorano\FIORAN~2\fmq)

Basic

This directory contains two sample programs which illustrate JMS Send-Receive mechanism using the FMQ C++ Runtime Library.

- **Sender.cpp** - Reads strings from standard input and sends the text messages on the queue "primaryQueue".
- **Receiver.cpp** - Implements a synchronous blocking receiver, which listens on the queue "primaryQueue", and prints the text of the received text messages on the console.

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. The %FMQ_DIR%\clients\cpp\native\scripts directory contains a script called cppclientbuild.bat which compiles the C++ program.
2. Run the Sender by executing the *Sender.exe* executable file.
3. Run the receiver by executing the *Receiver.exe* file.

Note: To run any of the C++ samples, ensure environment variable FMQ_DIR points to Fiorano's installation directory. (**Default**: C:\PROGRA~1\Fiorano\FIORAN~2\fmq)

Browser

This directory contains two sample programs which illustrate basic JMS Browser functionality using the FioranoMQ C++ Runtime Library.

- **QSender.cpp** - Reads strings from standard input and sends them on the queue "primaryqueue".
- **Browser.cpp** - Implements a browser, which is used to browse the messages on the queue "primaryqueue", and prints out the received messages.

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. For convenience, compiled versions of the sources are included in this directory. The %FMQ_DIR%\clients\cpp\native\scripts directory contains a script called cppclientbuild.bat which compiles the C++ program.
2. Run the Sender by executing the *QSender.exe* executable file. Send some messages before running the browser application
3. Run the browser by executing the *Browser.exe* file.

Note: To run any of the C++ samples, ensure that environment variable FMQ_DIR points to Fiorano's installation directory. (This defaults to C:\PROGRA~1\Fiorano\FIORAN~2\fmq)

HTTP

This directory contains two sample programs which illustrate the use of HTTP protocol for basic JMS PTP functionality using the FioranoMQ C++ Runtime Library.

- **QReceiver.cpp** - Receives messages asynchronously on primaryQueue. This program implements a synchronous listener to listen for messages published on the queue “primaryQueue”.
- **QSender.cpp** - Implements a client application publishing user specified data on primaryQueue. This program reads strings from standard input and publishes them on the Queue “primaryQueue”.

To run this sample using FioranoMQ, do the following:

1. Compile the source file. The %FMQ_DIR%\clients\cpp\native\scripts directory contains a script called cppclientbuild.bat which compiles the C++ program.
2. Run the HttpReceiver by executing the *QReceiver.exe* executable file.
3. Run the HttpSender by executing the *QSender.exe* executable file.

Note: To run any of the C++ samples, ensure that environment variable FMQ_DIR points to Fiorano installation directory. (**Default:** C:\PROGRA~1\Fiorano\FIORAN~2\fmq).

HTTPS

This directory contains four sample programs which illustrate the use of HTTPS protocol for basic JMS PTP functionality using the FioranoMQ C++ Runtime Library.

- **HReceiver.cpp** - Receives messages synchronously on primaryQueue. This program implements a synchronous listener to listen for messages published on the queue “primaryQueue”.
- **HSender.cpp** - Implements a client application publishing user specified data on primaryQueue. This program reads strings from standard input and publishes them on the Queue “primaryQueue”.

To run this sample using FioranoMQ, do the following:

1. Compile the source file. The %FMQ_DIR%\clients\cpp\native\scripts directory contains a script called cppclientbuild.bat which compiles the C++ program.
2. Run the HttpsReceiver by executing the *HReceiver.exe* executable file.
3. Run the HttpsSender by executing the *HSender.exe* executable file.

Note: To run any of the C++ samples, ensure that environment variable FMQ_DIR points to Fiorano installation directory (this defaults to C:\PROGRA~1\Fiorano\FIORAN~2\fmq).

MsgSel

This directory contains two sample programs which illustrate the use of message selectors using the FioranoMQ C++ Runtime Library.

- **QSelSender.cpp** - Selector sends messages with the string property "name" and an int property "value", set differently for 3 consecutive messages.
- **SelReceive.cpp** - Implements a synchronous listener, which listens on the queue "primaryqueue" for the messages which match the criteria specified in the message selector, and prints out the received messages.

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. The %FMQ_DIR%\clients\cpp\native\scripts directory contains a script called cppclientbuild.bat which compiles the C++ program.
2. Run the Sender by executing the *QSelSender.exe* executable file.
3. Run the synchronous receiver by executing the *QSelReceive.exe* file.

Note: To run any of the C++ samples, please ensure that environment variable FMQ_DIR points to Fiorano's installation directory. (**Default**: C:\PROGRA~1\Fiorano\FIORAN~2\fmq)

Mtptp

This directory contains one sample programs which illustrate basic JMS Sender/Receiver functionality using the FioranoMQ C++ Runtime Library multithreading support.

- **mtPtp.cpp** - The multithreaded version of basic PTP. Single Sender is created, sends 10 text messages on 'primaryQueue' and a single receiver blocking receive with timewait of 1 second reads the messages. Each executes on a separate thread. On receipt of 10 messages, the receiver notifies the main thread to end. Sender and Receiver threads are joined to the main thread.

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. The %FMQ_DIR%\clients\cpp\native\scripts directory contains a script called cppclientbuild.bat which compiles the C++ program.
2. Run the mtPtp by executing the *mtPtp.exe* executable file.

Note: To run any of the C++ samples, please ensure that environment variable FMQ_DIR points to Fiorano's installation directory. (**Default**: C:\PROGRA~1\Fiorano\FIORAN~2\fmq)

reqrep

This directory contains two folders Basic and timeout

basic

This directory contains two samples which illustrate JMS Request-Reply mechanism over Queues.

- **QueueRequestor.cpp** - Reads strings from standard input and sends the text messages on the queue "primaryQueue".
- **QueueReplier.cpp** - Implements an asynchronous listener, which listens on the queue "primaryQueue", and replies to the received message. The reply is sent on TemporaryQueue.

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. The %FMQ_DIR%\clients\cpp\native\scripts directory contains a script called cppclientbuild.bat which compiles the C++ program.
2. Run the Replier by executing the *QueueReplier.exe* executable file.
3. Run the requestor by executing the *QueueRequestor.exe* file.

Note: To run any of the C++ samples, please ensure that environment variable FMQ_DIR points to Fiorano's installation directory. (**Default**: C:\PROGRA~1\Fiorano\FIORAN~2\fmq)

TimedOut

This directory contains two sample programs which illustrate Timed Request-Reply mechanism over Queues using the FioranoMQ C++ Runtime Library.

- **TimedQueueRequestor.cpp** - Reads strings from standard input and sends the text messages on the queue "primaryQueue". The Requestor waits for a specified time for the reply. If the reply is not received within the stipulated time requestor times out.
- **TimedQueueReplier.cpp** - Implements an asynchronous listener, which listens on the queue "primaryQueue", and replies on a TemporaryQueue.

To run these samples using FioranoMQ, perform the following:

1. Compile each of the source files. The %FMQ_DIR%\clients\cpp\native\scripts directory contains a script called cppclientbuild.bat which compiles the C++ program.
2. Run the replier by executing the *QueueReplier.exe* executable file.
3. Run the timed requestor by executing the *TimedQueueRequestor.exe* file.

Note: To run any of the C++ samples, please ensure that environment variable FMQ_DIR points to Fiorano's installation directory. (**Default**: C:\PROGRA~1\Fiorano\FIORAN~2\fmq)

SSL

This directory contains two sample programs which illustrate the basic JMS Send/Receive functionality over Secure Socket Layer using the FioranoMQ C++ Runtime Library.

- **Sender.cpp** - Reads strings from standard input and sends them on the queue "primaryqueue".
- **Receiver.cpp** - Implements a synchronous listener, which listens on the queue "primaryqueue", and prints out the received messages.

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. The %FMQ_DIR%\clients\cpp\native\scripts directory contains a script called cppclientbuild.bat which compiles the C++ program.
2. Run the Sender by executing the *Sender.exe* executable file.
3. Run the synchronous receiver by executing the *Receiver.exe* file.

Note: To run any of the C++ samples, please ensure that environment variable FMQ_DIR points to Fiorano's installation directory. (**Default**: C:\PROGRA~1\Fiorano\FIORAN~2\fmq)

Transaction

This directory contains a sample programs which illustrate JMS Transaction functionality using the FioranoMQ C++ Runtime Library.

- **QTransaction.cpp** - Implements the sender and receiver, and uses the commit/rollback functionality to demonstrate JMS Transactions

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. The %FMQ_DIR%\clients\cpp\native\scripts directory contains a script called cppclientbuild.bat which compiles the C++ program.
2. Run the sample by executing the *QTransaction.exe* executable file. For proper results from the sample, ensure that there are no messages in the primaryQueue.

Note: To run any of the C++ samples, please ensure that environment variable FMQ_DIR points to Fiorano's installation directory. (**Default**: C:\PROGRA~1\Fiorano\FIORAN~2\fmq)

PubSub Samples

Admin

This directory contains one sample program which illustrates basic JMS Administration API functionality using the FioranoMQ C++ Runtime Library.

- **AdminTest.cpp** - Creates an Admin Connection with the MQServer and gets an MQAdminService object to create and delete Topics and TopicConnectionFacotries and retrieves information of users connected from the server.

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. The %FMQ_DIR%\clients\cpp\native\scripts directory contains a script called cppclientbuild.bat which compiles the C++ program.
2. Run the AdminTest by executing the *AdminTest.exe* executable file.

Note: To run any of the C++ samples, please ensure that environment variable FMQ_DIR points to Fiorano's installation directory. (**Default**: C:\PROGRA~1\Fiorano\FIORAN~2\fmq)

Basic

This directory contains two sample programs which illustrate basic JMS Publisher/Subscriber functionality using the FioranoMQ C++ Runtime Library.

- **Publisher.cpp** - Reads strings from standard input and sends them on the topic "primarytopic".
- **Subscriber.cpp** - Implements a synchronous listener, which listens on the topic "primarytopic", and prints out the received messages.

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. The %FMQ_DIR%\clients\cpp\native\scripts directory contains a script called cppclientbuild.bat which compiles the C++ program.
2. Run the Publisher by executing the *Publisher.exe* executable file.

3. Run the synchronous subscriber by executing the *Subscriber.exe* file.

Note: To run any of the C++ samples, please ensure that environment variable FMQ_DIR points to Fiorano's installation directory. (**Default**: C:\PROGRA~1\Fiorano\FIORAN~2\fmq)

Dursub

This directory contains two sample programs which illustrate basic JMS DurableSubscriber functionality using the FioranoMQ C++ Runtime Library.

- **DurPublisher.cpp -** Reads strings from standard input and publishes PERSISTENT messages on the topic "primarytopic".
- **DurSubscriber.cpp -** Implements a durable subscriber using the client ID "DS_Client_1" and durable subscriber name "Sample_Durable_Subscriber", listening on the topic "primarytopic".

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. The %FMQ_DIR%\clients\cpp\native\scripts directory contains a script called *cppclientbuild.bat* which compiles the C++ program.
2. Start the DurableSubscriber program first, so that the subscriber can register with the FioranoMQ Server.
3. Next, start the Publisher_d program. When the program comes up, type in a few strings, pressing the Enter key after each string. The string is published and is received by the Durable Subscriber started in step (2) above.
4. Now, shut down the Durable Subscriber, but keep typing in messages into the Publisher program. These messages are automatically stored by the FioranoMQ Server, since a Durable Subscriber was previously registered on the topic to which the messages are being published.
5. After a while, restart the DurableSubscriber program. On restart, you would find that all messages that were published during the time that the durable subscriber was down are now made available to the subscriber.
6. Repeat steps (4) and (5) over. Each time, you would find that all messages published during the time that the Subscriber is down are immediately made available to the Subscriber when it restarts.

Note: To run any of the C++ samples, please ensure that environment variable FMQ_DIR points to Fiorano's installation directory. (**Default**: C:\PROGRA~1\Fiorano\FIORAN~2\fmq)

HTTP

This directory contains four sample programs which illustrate the use of HTTP protocol for basic JMS PubSub functionality using the FioranoMQ C++ Runtime Library.

- **Subscriber.cpp** - Receives messages synchronously published on "primaryTopic". This program implements an synchronous listener to listen for messages published on “primaryTopic".
- **Publisher.cpp** - Implements a client application publishing user specified data on "primaryTopic". This program reads strings from standard input and publishes them on “primaryTopic”.

To run this sample using FioranoMQ, do the following:

1. Compile the source file. The %FMQ_DIR%\clients\cpp\native\scripts directory contains a script called cppclientbuild.bat which compiles the C++ program.
2. Run the HttpSubscriber by executing the *Subscriber.exe* executable file.
3. Run the HttpPublisher by executing the *Publisher.exe* executable file.

Note: To run any of the C++ samples, please ensure that environment variable FMQ_DIR points to Fiorano installation directory. (This defaults to C:\PROGRA~1\Fiorano\FIORAN~2\fmq).

HTTPS

This directory contains two sample programs which illustrate basic JMS Publisher/Subscriber functionality using the Fio C++ Runtime Library.

- **HPublisher.cpp -** Reads strings from standard input and sends them on the topic "primarytopic".
- **HSubscriber.cpp** - Implements a synchronous listener, which listens on the topic "primarytopic", and prints out the received messages.

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. The %FMQ_DIR%\clients\cpp\native\scripts directory contains a script called cppclientbuild.bat which compiles the C++ program.
2. Run the Publisher by executing the *Publisher.exe* executable file.
3. Run the synchronous subscriber by executing the *Subscriber.exe* file.

Note: To run any of the C++ samples, please ensure that environment variable FMQ_DIR points to Fiorano's installation directory. (**Default**: C:\PROGRA~1\Fiorano\FIORAN~2\fmq)

Msgsel

This directory contains two sample programs which illustrate the use of message selectors using the FioranoMQ C++ Runtime Library.

- **SelSend.cpp -** Selector sends messages with the string property "name" and an int property "value", set differently for 3 consecutive messages.
- **SelRecv.cpp** - Implements a synchronous listener, which listens on the topic "primarytopic" for the messages which match the criteria specified in the message selector, and prints out the received messages.

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. The %FMQ_DIR%\clients\cpp\native\scripts directory contains a script called cppclientbuild.bat which compiles the C++ program.
2. Run the Sender by executing the *SelSend.exe* executable file.
3. Run the synchronous receiver by executing the *SelRecv.exe* file.

Note: To run any of the C++ samples, please ensure that environment variable FMQ_DIR points to Fiorano's installation directory. (**Default**: C:\PROGRA~1\Fiorano\FIORAN~2\fmq)

Mtpubsub

This directory contains one sample programs which illustrate basic JMS Publish/Subscribe functionality using the FioranoMQ C++ Runtime Library multithreading support.

- **mtPubSub.cpp** - The multithreaded version of basic PubSub. Single Publisher is created, publishes 10 text messages on 'primaryTopic' and a single subscriber blocking receive with timewait of 1 second reads the messages. Each executes on a separate thread. On receipt of 10 messages, the subscriber notifies the main thread to end. Publisher and Subscriber threads are joined to the main thread.

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. The %FMQ_DIR%\clients\cpp\native\scripts directory contains a script called cppclientbuild.bat which compiles the C++ program.
2. Run the mtPubSub by executing the *mtPubSub.exe* executable file.

Note: To run any of the C++ samples, please ensure that environment variable FMQ_DIR points to Fiorano's installation directory. (**Default**: C:\PROGRA~1\Fiorano\FIORAN~2\fmq)

Reqrep

This directory contains two folders basic and timeout.

Basic

This directory contains two sample programs which illustrate JMS Request-Reply mechanism over Topics using the FioranoMQ C++ Runtime Library.

- **TopicRequestor.cpp -** Reads strings from standard input and sends the text messages on the topic "primaryTopic".
- **TopicReplier.cpp** - Implements an asynchronous listener, which listens on the topic "primaryTopic", and replies to the received. The reply is sent on TemporaryTopic.

To run these samples using FioranoMQ, perform the following:

1. Compile each of the source files. The %FMQ_DIR%\clients\cpp\native\scripts directory contains a script called cppclientbuild.bat which compiles the C++ program.
2. Run the Replier by executing the *TopicReplier.exe* executable file.
3. Run the requestor by executing the *TopicRequestor.exe* executable file.

TimedOut

This directory contains two sample programs which illustrate Timed Request-Reply mechanism over Topics using the FioranoMQ C++ Runtime Library.

- **TimedTopicRequestor.cpp -** Reads strings from standard input and sends the text messages on the topic "primaryTopic". The Requestor waits for a specified time for the reply. If the reply is not received within the stipulated time requestor times out.
- **TopicReplier.cpp** - Implements an asynchronous listener, which listens on the topic "primaryTopic", and replies on a TemporaryTopic.

To run these samples using FioranoMQ, perform the following:

1. Compile each of the source files. The %FMQ_DIR%\clients\cpp\native\scripts directory contains a script called cppclientbuild.bat which compiles the C++ program.
2. Run the replier by executing the *TopicReplier.exe* executable file.
3. Run the timed requestor by executing the *TimedTopicRequestor.exe* file.

Note: To run any of the C++ samples, please ensure that environment variable FMQ_DIR points to Fiorano's installation directory. (**Default**: C:\PROGRA~1\Fiorano\FIORAN~2\fmq)

SSL

This directory contains two sample programs which illustrate basic JMS Publisher/Subscriber functionality using the FioranoMQ C++ Runtime Library.

- **Publisher.cpp** - Reads strings from standard input and sends them on the topic "primarytopic".
- **Subscriber.cpp** - Implements a synchronous listener, which listens on the topic "primarytopic", and prints out the received messages.

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. The %FMQ_DIR%\clients\cpp\native\scripts directory contains a script called cppclientbuild.bat which compiles the C++ program.
2. Run the Publisher by executing the *Publisher.exe* executable file.
3. Run the asynchronous subscriber by executing the *Subscriber.exe* file.

Note: To run any of the C++ samples, please ensure that environment variable FMQ_DIR points to Fiorano's installation directory. (**Default**: C:\PROGRA~1\Fiorano\FIORAN~2\fmq)

Transaction

This directory contains a sample programs which illustrate JMS Transaction functionality using the FioranoMQ C++ Runtime Library.

- **Transaction.cpp** - Implements the sender and receiver, and uses the commit/rollback functionality to demonstrate JMS Transactions

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. The %FMQ_DIR%\clients\cpp\native\scripts directory contains a script called cppclientbuild.bat which compiles the C++ program.
2. Run the sample by executing the *Transaction.exe* executable file. For proper results from the sample, ensure that there are no messages in the primaryTopic

Note: To run any of the CPP samples, please ensure that environment variable FMQ_DIR points to Fiorano's installation directory. (**Default**: C:\PROGRA~1\Fiorano\FIORAN~2\fmq)

Chapter 8: Frequently Asked Questions

Question 1: What is a FioranoMQ client?

Answer: The FioranoMQ clients are thin RTL implementations that help applications written in various supported languages for different supported platforms to communicate seamlessly with each other making use of the robust Fiorano JMS Server.

Question 2: What is the FioranoMQ native C++RTL?

Answer: The native C++RTL client is the C++ language version of the RTL implemented in native form. The stress on native implementation is for some customers who do not want the runtime to have any dependency with the JRE.

Question 3: What are the other languages supported by FioranoMQ client RTLs?

Answer: Currently Fiorano provides java, C, C++ and C# implementations of the FioranoMQ client.

Question 4: Do all versions of the RTL support the same set of functionalities?

Answer: Essentially the core functionality(pub/sub and ptp routines) is available through all the versions of the client RTLs. Administration of the MQServer using C++ APIs have been implemented. Advanced features similar to Java RTL like client side persistence (CSP & Durable Subscribers), auto reconnection to server logic (EnableAutoRevalidation) are also available now.

Question 5: What are the platforms on which the native C++RTL has been tested?

Answer: The current version (1.0) of C++RTL has been tested to run on Microsoft Windows 2000, Microsoft Windows XP and Sun Solaris Operating environment, Version 7.

Question 6: What are the supported transport protocols?

Answer: The native C++RTL supports TCP, HTTP, HTTPS and PHAOS SSL protocols for transport.

Question 7: What is .NET framework?

Answer: Microsoft .NET Framework is a platform for building, deploying, and running Web Services and applications. It provides a highly productive, standards-based, multi-language environment for integrating existing investments with next-generation applications and services and addresses the challenges of deployment and operation of Internet-scale applications. The .NET Framework contains the following components:

- A common language runtime
- A hierarchical set of unified class libraries
- A componentized version of Active Server Pages called ASP.NET

Question 8: What is GC?

Answer: Garbage collection is a mechanism that allows the computer to detect when an object can no longer be accessed. It then automatically releases the memory used by that object (as well as calling a clean-up routine, called a "finalizer," which is written by the user). Some garbage collectors, like the one used by .NET, compact memory and therefore decrease your program's working set.

Question 9: What is the FioranoMQ native C++RTL .NET version?

Answer: The .NET version of FioranoMQ C++RTL is the C++ language version of the RTL implemented for the .NET platform. Using the FioranoMQ C++RTL (.NET version), .NET applications can now seamlessly communicate with each other and with the whole range of applications written in any supported language on any supported platform.

Question 10: How can I solve the error "Application failed to start because pthreadVC.dll was not found"?

Answer: This error occurs when the pthreadVC library was not found in the system path. *pthreadVC.dll* can be found in the clients\pthread\lib directory of FioranoMQ installation. Copying this file to the system path solves the issue.

Question 11: How can I solve this error "pthread missing" while running the samples?

Answer: When C++ samples are compiled the executables are created in the current directory. Before running them, please ensure that the pthread library is found in the system path.