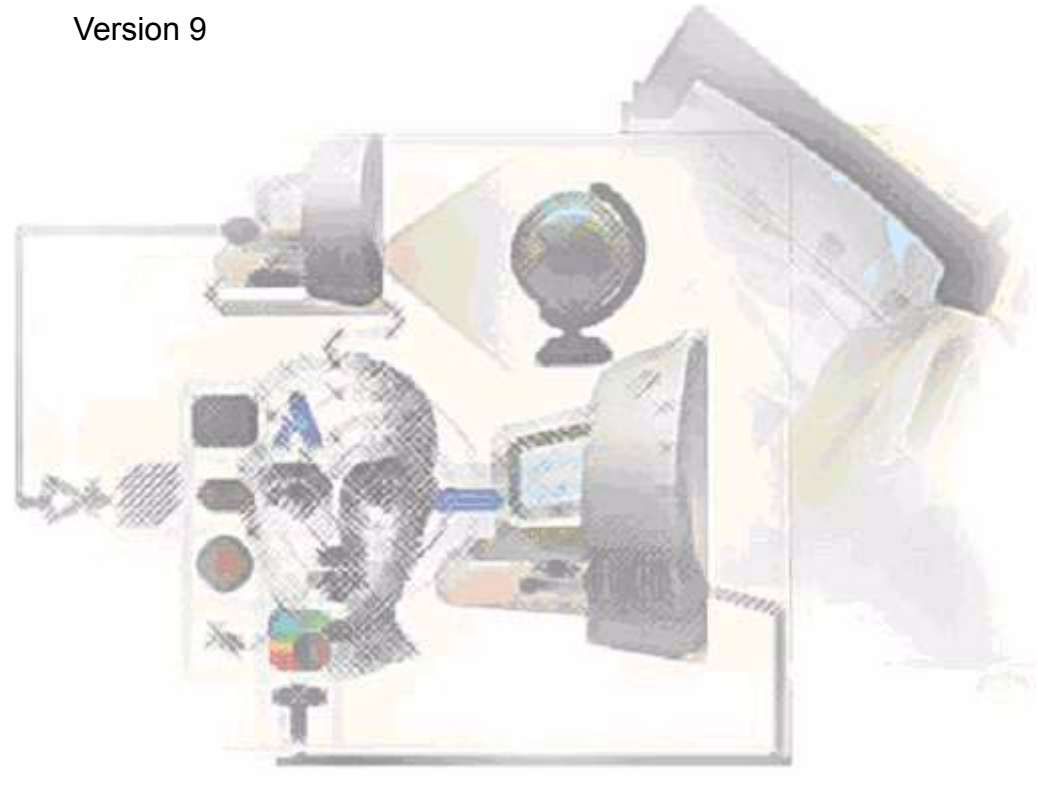
Fiorano Software, Inc.

FioranoMQ 9 – Messaging Server

C# RTL Guide

Version 9



Fiorano Software, Inc.
718, University Avenue, Suite 212,
Los Gatos, California 95032 U.S.A.
Tel 1.408.354.3210
e-mail: info@fiorano.com

Contents

Contents **3**

Chapter 1:Preface 11

Audience 11
Overview of the Guide 11
Introduction to FMQ C#RTL 11
Typographical Conventions 12
Related Documentation 12

Chapter 2:Datatypes and Constants 15

Naming convention 15

Chapter 3:Error Handling 17

Chapter 4:Function Reference 19

Helpers 20
CsHashTable 20
Constructor 20
Put 20
Get 20
RemoveElement 21
HashTableEnumerator 21
hasMoreElemets 21
nextKeyElement 22
nextValueElement 22
JMS Interfaces 23
CsByteMessage 23
getBodyLength 23
readBoolean 23
readByte 24
readBytes 24
readChar 25
readDouble 25
readFloat 26
readInt 26
readLong 26
readShort 27
readUnsignedByte 27
readUnsignedShort 28
readUTF 28
reset 29
writeBoolean 29
writeByte 29
writeBytes 30
writeChar 30

writeDouble . . . 31
writeFloat . . . 31
writeInt . . . 32
writeLong . . . 32
writeShort . . . 33
writeUTF . . . 33
CsConnection . . . 34
CsConnectionConsumer . . . 34
CsConnectionFactory . . . 34
CsConnectionMetaData . . . 34
CsDestination . . . 35
CsException . . . 35
checkForException . . . 35
CJMSException . . . 35
Constructor . . . 35
Constructor . . . 36
checkForException . . . 36
printStackTrace . . . 37
getErrorCode . . . 37
getLinkedException . . . 37
CsExceptionListener . . . 37
onException . . . 38
CsJMSException . . . 38
Constructor . . . 38
Constructor . . . 38
checkForException . . . 39
printStackTrace . . . 39
getErrorCode . . . 39
getLinkedException . . . 40
CsMapMessage . . . 40
getBoolean . . . 40
getByte . . . 41
getChar . . . 41
getDouble . . . 42
getFloat . . . 42
getInt . . . 43
getLong . . . 43
getMapNames . . . 44
getShort . . . 44
getString . . . 45
itemExists . . . 45
setBoolean . . . 46
setByte . . . 46
setBytes . . . 47
setBytes . . . 47
setChar . . . 48
setDouble . . . 48
setFloat . . . 49

setInt 49
setLong 50
setShort 50
setString 51
CsMessage 51
acknowledge 51
clearBody 52
clearProperties 52
getBooleanProperty 53
getByteProperty 53
getDoubleProperty 54
getFloatProperty 54
getIntProperty 55
getJMSCorrelationID 55
getJMSCorrelationIDAsBytes 56
getJMSDeliveryMode 56
getJMSDestination 57
getJMSExpiration 57
getJMSMessageID 58
getJMSPriority 59
getJMSRedelivered 59
getJMSReplyTo 60
getJMSTimestamp 60
getJMSType 61
getLongProperty 61
getObjectProperty 62
getShortProperty 62
getStringProperty 63
propertyExists 63
setBooleanProperty 64
setByteProperty 64
setDoubleProperty 65
setFloatProperty 65
setIntProperty 66
setJMSCorrelationID 66
setJMSDeliveryMode 67
setJMSDestination 67
setJMSExpiration 68
setJMSMessageID 68
setJMSPriority 69
setJMSRedelivered 69
setJMSReplyTo 70
setJMSTimestamp 70
setJMSType 71
setLongProperty 71
setObjectProperty 72
setshortProperty 72
setStringProperty 73

CsMessageConsumer . . . 73
CsMessageListener . . . 73
CsServerSession . . . 74
CsServerSessionPool . . . 74
CsSession . . . 74
CsStreamMessage . . . 75
readBoolean . . . 75
readByte . . . 76
readBytes . . . 76
readChar . . . 77
readDouble . . . 77
readFloat . . . 77
readInt . . . 78
readLong . . . 78
readShort . . . 79
readString . . . 79
reset . . . 80
writeBoolean . . . 80
writeByte . . . 80
writeBytes . . . 81
writeBytes . . . 81
writeChar . . . 82
writeDouble . . . 82
writeFloat . . . 83
writeInt . . . 83
writeLong . . . 84
writeShort . . . 84
writeString . . . 84
CsTextMessage . . . 85
getText . . . 85
setText . . . 86
Naming and Lookup (JNDI) . . . 87
CsInitialContext . . . 87
Constructor . . . 87
Lookup . . . 87
PTP . . . 89
CsQueue . . . 89
getQueueName . . . 89
toString . . . 89
CsQueueConnection . . . 90
createQueueSession . . . 90
close . . . 91
getClientID . . . 91
setClientID . . . 92
start . . . 92
stop . . . 93
getExceptionListener . . . 93

setExceptionListener 94
CsQueueConnectionFactory 94
createQueueConnection 94
createQueueConnection 95
CsQueueReceiver 95
getQueue 96
close 96
getMessageListener 97
getMessageSelector 97
receive 97
recieve 98
reveiveNoWait 98
set 99
CsQueueRequestor 99
close 100
request 100
request 101
QueueBrowser 101
close 101
getMessageSelector 102
getQueue 102
CsQueueSender 103
getQueue 103
close 103
getDeliveryMode 104
getDestination 104
getDisableMessageID 104
getDisableMessageTimestamp 105
getPriority 105
getTimeToLive 106
send 106
send 107
send 107
send 108
setDeliveryMode 108
setDisableMessageID 109
setDisableMessageTimeStamp 109
setPriority 110
setTimeToLive 110
CsQueueSession 111
close 111
commit 111
createBrowser 112
createBrowser 112
createBytesMesage 113
createMapMessage 113
createQueue 114
createStreamMessage 114

createTemporaryQueue 115
createTextMessage 115
createTextMessage 115
getMessageListener 116
recover 116
rollback 117
run 117
setMessageListener 117
CsTemporaryQueue 118
remove 118
Publish/Subscribe 119
CsTemporaryTopic 119
remove 119
CsTopic 119
getTopicName 120
toString 120
CsTopicConnection 120
createTopicSession 121
close 121
getClientID 122
setClientID 122
start 122
stop 123
getExceptionListener 123
setExceptionListener 124
CsTopicConnectionFactory 124
createTopicConnection 124
createTopicConnection 125
CsTopicPublisher 125
getTopic 126
publish 126
publish 126
publish 127
publish 128
close 128
getDeliveryMode 129
getDestination 129
getDisableMessageID 130
getDisableMesageTimestamp 130
getPriority 130
getTimeToLive 131
send 131
send 132
send 132
send 133
setDeliveryMode 134
setDisableMessageID 134
setDisableMessageTimestamp 135

setPriority . . . 135
setTimeToLive . . . 136
CsTopicRequestor . . . 136
close . . . 136
request . . . 137
request . . . 137
CsTopicSession . . . 138
createPublisher . . . 138
createSubscriber . . . 139
createSubscriber . . . 139
close . . . 140
commit . . . 140
createBytesMessage . . . 140
createDurableSubscriber . . . 141
createDurableSubscriber . . . 141
createMapMessage . . . 142
createStreamMessage . . . 143
createTemporaryQueue . . . 143
createTextMessage . . . 143
createTextMessage . . . 144
createTopic . . . 144
getMessageListener . . . 145
recover . . . 145
rollback . . . 145
setMessageListener . . . 146
unsubscribe . . . 146
CsTopicSubscriber . . . 147
close . . . 147
getMessageListener . . . 147
getMessageSelector . . . 148
receive . . . 148
receive . . . 149
receiveNoWait . . . 149
setMessageListener . . . 150
getNoLocal . . . 150
getTopic . . . 151

Chapter 5:Using the Samples Programs . . . 153

Organization of Samples Provided . . . 153
Compiling and Running Samples . . . 153
Limitations of C#RTL . . . 154

Chapter 6:Frequently Asked Questions . . . 155

1

Preface

Audience

This guide is designed to assist developers working on the .Net technology to understand and use FioranoMQ C#RTL library. The developer must have a fair knowledge of the C# programming language.

Overview of the Guide

This guide contains the following contents.

- An introduction to FioranoMQ C# RunTime library.
- Detailed description of the APIs included in this library.
- Details on compiling and running C#RTL sample applications.
- Frequently Asked Questions (FAQ).

Introduction to FMQ C#RTL

The C#RTL allows C# applications to communicate seamlessly with C, C++ and java programs. This library helps in bridging the gap between J2EE and .NET domain and enables .NET applications to directly talk to the Java server. This version of C#RTL supports secure and non-secure TCP and HTTP connections on Win32 platform for Point-to-Point and Publish/Subscribe communication models.

The C#RTL has been designed to provide maximum conformance to JMS specifications. All public APIs have similar signature as the corresponding Java APIs specified by JMS. All classes have a similar naming convention. Exception handling is also similar to that specified by JMS with all APIs throwing a CJMSException with a specific error Code and/or error description.

☞ For more information on JMS specifications, please refer to Java Message Service Specification on the Sun Microsystems website.

Typographical Conventions

To locate and interpret information easily, the manual uses consistent visual cues, and standard text formats. The Table below lists the various conventions used in the manual.

Conventions	Description
Note	Additional/important information.
See Also	Reference to important information located elsewhere in the documentation framework.
Warning	Information regarding a potential aberration if a component is used incorrectly.
1. Numbered List	Instructions that must be followed in a sequential order.
■ Bulleted List	A list where sequence is immaterial.
C:\Programs\FioranoMQ	Directory paths
start()	Class names, methods, and interfaces.
`public class ChatService extends FioranoBISService`	Code samples
`fiorano.FioranoBIS.common.*`	This style is used to highlight: 1. Program code within paragraphs 2. Values to be entered by users in a procedure

TABLE 1 Typographical Conventions

Related Documentation

For complete information on the available RunTime libraries, we strongly recommend that you go through the entire range of FioranoMQ RTL Documentation.

Document Name	Description
C Runtime Library Guide	Contains the description of the FioranoMQ C Runtime APIs.
Native C ++ Runtime Library Guide	Contains a description of the FioranoMQ native C++ Runtime APIs.

Document Name	Description
JNI-based C ++ Runtime Library Guide	Provides an introduction toFioranoMQ JNI based C++ Runtime APIs
JavaDocs	Contains "javadoc" style documentation on all public Fiorano classes.

TABLE 2 Related Documentation

2

Datatypes and Constants

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

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	Cs<JMSClass name>	`CsTopicPublisher(JMS:Top-icPublisher)`
Constant	same as in JMS spec	`NON_PERSISTENT`
Function	same as in JMS spec	`publish`

Error Handling

3

The C#RTL uses exceptions to provide error handling. In event of an error in the C#RTL layer, CJMSException with a specific Error Code 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 exception that occurred in one thread doesn't interfere 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();
}
```

For more information, read the CJMSException section.

4

Function Reference

This chapter 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 chapter is summarized as follows.

Contents
Helpers
JMS Interfaces
Naming and Lookup (JNDI)
PTP
Publisher/Subscriber

Helpers

CsHashTable

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

Version

1.0

Constructor

Description

Constructs a new, empty hashtable.

Declaration

```
CsHashTable()
```

Put

Description

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

Version

1.0

Declaration

```
object Put(string key, object value)
```

Get

Description

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

Version

1.0

Declaration

```
object Get(string key)
```

RemoveElement

Description

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

Version

1.0

Declaration

```
object RemoveElement(string key)
```

HashTableEnumerator

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

Version

1.0

hasMoreElemets

Description

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

Version

1.0

Declaration

```
bool hasMoreElements();
```

nextKeyElement

Description

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

Version

1.0

Declaration

```
string nextKeyElement();
```

Returns

Returns a string containing the next key value

nextValueElement

Description

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

Version

1.0

Declaration

```
object nextValueElement();
```

Returns

Returns the next value element

JMS Interfaces

CsByteMessage

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

Derives

CsMessage

Version

1.0

getBodyLength

Description

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 returnedis the entire length of the message body, regardless of where the pointer for reading the message is currently located.

Declaration

```
long getBodyLength()
```

Returns

Number of bytes in the message.

Throws

`CJMSException`

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

readBoolean

Description

Reads a boolean from the bytes message stream.

Declaration

```
bool readBoolean();
```

Returns

The boolean value read

Throws

`CJMSException`

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

readByte

Description

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

Declaration

```
byte readByte()
```

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

Description

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

```
int readBytes(byte[] value, int length);
```

Parameters

`value`

The buffer into which the data is read.

`length`

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

Description

Reads a Unicode character value from the bytes message stream.

Declaration

```
char readChar();
```

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

Description

Reads a double from the bytes message stream.

Declaration

```
double readDouble();
```

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

Description

Reads a float from the bytes message stream.

Declaration

```
float readFloat();
```

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

Description

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

Declaration

```
int readInt();
```

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

Description

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

Declaration

```
long readLong()
```

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

Description

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

Declaration

```
short readShort();
```

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

Description

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

Declaration

```
int readUnsignedByte();
```

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

Description

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

Declaration

```
int readUnsignedShort();
```

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

Description

Reads a string 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

```
string readUTF()
```

Returns

A Unicode string from the bytes message stream

Throws

`CJMSException`

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

reset

Description

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

Declaration

```
void reset();
```

Throws

`CJMSException`

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

writeBoolean

Description

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

Declaration

```
void writeBoolean(bool value);
```

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

Description

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

Declaration

```
void writeByte(byte value);
```

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

Description

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

Declaration

```
void writeBytes(byte[] value, int offset, int length);
```

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

Description

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

Declaration

```
void writeChar(char value);
```

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

Description

Reads a double from the bytes message stream.

Declaration

```
void writeDouble(double value);
```

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

Description

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(float value);
```

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

Description

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

Declaration

```
void writeInt(int value);
```

Parameters

`value`

The int to be written

Throws

`CJMSException`

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

writeLong

Description

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

Declaration

```
void writeLong(long value);
```

Parameters

`value`

The long to be written

Throws

`CJMSException`

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

writeShort

Description

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

Declaration

```
void writeShort(short value);
```

Parameters

`value`

The short to be written

Throws

`CJMSException`

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

writeUTF

Description

Writes a string 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(string value);
```

Parameters

`value`

The String value to be written

Throws

`CJMSException`

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

CsConnection

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

Subclasses

`CsQueueConnection CsTopicConnection`

Version

1.0

CsConnectionConsumer

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

Version

1.0

CsConnectionFactory

A CsConnectionFactory 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.

Sub Classes

`CsTopicConnectionFactory CsQueueConnectionFactory`

Version

1.0

CsConnectionMetaData

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

Subclasses

`CsTopicSubscriber CsQueueReceiver`

Version

1.0

CsDestination

A CsDestination object encapsulates a provider-specific address.

Subclasses

`CsTopic, CsQueue, CsTemporaryTopic, CsTemporaryQueue`

Version

1.0

CsException

This is the root class of all exceptions.

Version

1.0

public

checkForException

Description

Utility Function that throws CJMSException if an exception had occurred.

Declaration

```
static void checkForException();
```

Version

1.0

CJMSException

This object is a part of the C++ RunTimeLibrary.

Constructor

Description

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

Declaration

```
CJMSException(mqstring errCode);
```

Parameters

`errCode`

The error code that identifies the error

Constructor

Description

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

Declaration

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

Parameters

`errCode`

The error code that identifies the error

`errDesc`

Description of the error

checkForException

Description

Utility Function that throws CJMSException if an exception had occurred.

Declaration

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

Parameters

`errCode`

The error code that identifies the error

`errDesc`

Description of the error

printStackTrace

Description

Prints the Function stack trace on the console

Declaration

```
void printStackTrace();
```

getErrorCode

Description

Gets the vendor-specific error code.

Declaration

```
const mqstring getErrorCode();
```

Returns

The vendor-specific error code

getLinkedException

Description

Gets the exception linked to this one

Declaration

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

Returns

The linked exception

CsExceptionListener

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

Version

1.0

onException

Description

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

Declaration

```
abstract void onException(CException msg);
```

Parameters

`msg`

Message handle

Version

1.0

CsJMSException

Constructor

Description

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

Declaration

```
CJMSException(string errCode);
```

Parameters

`errCode`

The error code that identifies the error

Constructor

Description

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

Declaration

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

Parameters

`errCode`

The error code that identifies the error

`errDesc`

Description of the error

checkForException

Description

Utility Function that throws CJMSException if an exception had occurred.

Declaration

```
static void checkForException(string errCode, string errDesc);
```

Parameters

`errCode`

The error code that identifies the error

`errDesc`

Description of the error

printStackTrace

Description

Prints the Function stack trace on the console

Declaration

```
void printStackTrace();
```

getErrorCode

Description

Gets the vendor-specific error code.

Declaration

```
const string getErrorCode();
```

Returns

The vendor-specific error code

getLinkedException

Description

Gets the exception linked to this one

Declaration

```
CJMSException getLinkedException();
```

Returns

The linked exception

CsMapMessage

The CsMapMessage 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 string. The entries can be accessed sequentially or randomly by name. CsMapMessage inherits from the CsMessage interface and adds a message body that contains a Map.

Derives

```
CsMessage
```

Version

1.0

getBoolean

Returns the boolean value with the specified name.

Declaration

```
boolean getBoolean(string name);
```

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

Description

Returns the byte value with the specified name.

Declaration

```
byte getByte(string name);
```

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

Description

Returns the Unicode character value with the specified name.

Declaration

```
char getChar(string name);
```

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

Description

Returns the double value with the specified name.

Declaration

```
double getDouble(string name);
```

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

Description

Returns the float value with the specified name.

Declaration

```
float getFloat(string name);
```

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

Description

Returns the int value with the specified name.

Declaration

```
int getInt(string name);
```

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

Description

Returns the long value with the specified name.

Declaration

```
long getLong(string name);
```

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

Description

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

Declaration

```
CHashTableEnumerator *getMapNames();
```

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

Description

Returns the short value with the specified name.

Declaration

```
short getShort(string name);
```

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

Description

Returns the String value with the specified name.

Declaration

```
string getString(java.lang.String name);
```

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

Description

Indicates whether an item exists in this MapMessage object.

Declaration

```
boolean itemExists(string name);
```

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

Description

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

Declaration

```
void setBoolean(string name, boolean value);
```

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

Description

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

Declaration

```
void setByte(string name, byte value);
```

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

Description

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

Declaration

```
void setBytes(string name, byteArray value);
```

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

Description

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

Declaration

```
void setBytes(string name, byteArray value, int offset, int
length);
```

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`

setChar

Description

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

Declaration

```
void setChar(string name, char value);
```

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

setDouble

Description

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

Declaration

```
void setDouble(string name, double value);
```

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

Description

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

Declaration

```
void setFloat(string name, float value);
```

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

Description

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

Declaration

```
void setInt(string name, int value);
```

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

Description

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

Declaration

```
void setLong(string name, long value);
```

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

Description

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

Declaration

```
void setShort(string name, short value);
```

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

Description

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

Declaration

```
void setString(string name, string value);
```

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.

CsMessage

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

Subclasses

`CsTextMessage, CsBytesMessage, CsMapMessage, CsObjectMessage, CsStreamMessage`

Version

1.0

acknowledge

Description

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 to. 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 acknowl-

edge 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();
```

Throws

`CJMSException`

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

clearBody

Description

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();
```

Throws

`CJMSException`

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

clearProperties

Description

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

Declaration

```
void clearProperties();
```

Throws

`CJMSException`

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

getBooleanProperty

Description

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

Declaration

```
bool getBooleanProperty(string name);
```

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

Description

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

Declaration

```
byte getByteProperty(string name);
```

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

Description

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

Declaration

```
double getDoubleProperty(string name);
```

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

Description

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

Declaration

```
float getFloatProperty(string name);
```

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

```
int getIntProperty(string name);
```

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

Description

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

```
string getJMSCorrelationID();
```

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

Description

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

Declaration

```
string getJMSCorrelationIDAsBytes();
```

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.

getJMSDeliveryMode

Description

Gets the DeliveryMode value specified for this message.

Declaration

```
int getJMSDeliveryMode();
```

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

Description

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

```
CsDestination getJMSDestination();
```

Returns

The destination of this message

Throws

`CJMSException`

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

getJMSExpiration

Description

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

```
long getJMSExpiration();
```

Returns

The time when 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

Description

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 JMSprovider 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

```
string getJMSMessageID();
```

Returns

The message ID

Throws

`CJMSException`

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

getJMSPriority

Description

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

```
int getJMSPriority();
```

Returns

The default message priority

Throws

`CJMSException`

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

getJMSRedelivered

Description

Gets an indication of whether this message is being redelivered. If a client receives a message with the MSRedelivered 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

```
bool getJMSRedelivered();
```

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

Description

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

Declaration

```
CsDestination getJMSReplyTo();
```

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

Description

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

Declaration

```
long getJMSTimestamp();
```

Returns

Returns the timestamp of the message

Throws

`CJMSException`

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

getJMSType

Description

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

Declaration

```
string getJMSType();
```

Returns

The message type

Throws

`CJMSException`

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

getLongProperty

Description

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

Declaration

```
long getLongProperty(string propName);
```

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

Description

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

```
object getObjectProperty(string propName);
```

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.

getShortProperty

Description

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

Declaration

```
short getShortProperty(string propName)
```

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

Description

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

Declaration

```
string getStringProperty(string propName);
```

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.

propertyExists

Description

Indicates whether a property value exists.

Declaration

```
bool propertyExists(string propName);
```

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

Description

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

Declaration

```
void setBooleanProperty(string propName, bool value);
```

Parameters

`PropName`

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

Description

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

Declaration

```
void setByteProperty(string propName, byte value)
```

Parameters

`PropName`

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

Description

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

Declaration

```
void setDoubleProperty(string propName, double value);
```

Parameters

`PropName`

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

Description

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

Declaration

```
void setFloatProperty(string propName, float value);
```

Parameters

`PropName`

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

Description

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

Declaration

```
void setIntProperty(string propName, int value);
```

Parameters

`PropName`

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

Description

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(string corrID);
```

Parameters

`correlationID`

The message ID of a message being referred to

Throws

`CJMSException`

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

setJMSDeliveryMode

Description

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(int deliveryMode);
```

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

Description

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(CsDestination dest);
```

Parameters

`destination`

The destination for this message

Throws

`CJMSException`

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

setJMSExpiration

Description

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(long expiration);
```

Parameters

`expiration`

The message's expiration time

Throws

`CJMSException`

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

setJMSMessageID

Description

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(string msgID);
```

Parameters

`id`

The ID of the message

Throws

`CJMSException`

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

setJMSPriority

Description

Sets the priority level 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 setJMSPriority(int priority);
```

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

Description

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(bool redelivered);
```

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

Description

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(CsDestination dest);
```

Parameters

`dest`

Destination to which to send a response to this message

Throws

`CJMSException`

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

setJMSTimestamp

Description

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(long timestamp);
```

Parameters

`timestamp`

The timestamp for this message

Throws

`CJMSException`

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

setJMSType

Description

Sets the message type. Some JMS providers use a message repository thatcontains 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(string type);
```

Parameters

`type`

The message type

Throws

`CJMSException`

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

setLongProperty

Description

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

Declaration

```
void setLongProperty(string propName, long value);
```

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

Description

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(string propName, object value, int size)
```

Parameters

`propName`

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

Description

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

Declaration

```
void setShortProperty(string propName, short value)
```

Parameters

`propName`

The name of the short property

`value`

The short property value to set.

Throws

`CJMSException`

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

setStringProperty

Description

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

Declaration

```
void setStringProperty(string propName, string value);
```

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.

CsMessageConsumer

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

Subclasses

`CsTopicSubscriber CsQueueReceiver`

Version

1.0

CsMessageListener

A CsMessageListener object is used to receive asynchronously delivered messages.

Version

1.0

Description

Passes a message to the listener.

Declaration

```
abstract void onMessage(CsMessage msg);
```

Parameters

`message`

The message passed to the listener

CsServerSession

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

Version

1.0

CsServerSessionPool

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

Subclasses

`CsTopicSession CsQueueSession`

Version

1.0

CsSession

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

Subclasses

`CsTopicSession CsQueueSession`

Version

1.0

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
```

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

```
static const int CCLIENT_ACKNOWLEDGE
```

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

```
static const int CDUPS_OK_ACKNOWLEDGE
```

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

```
static const int CSESSION_TRANSACTED
```

CsStreamMessage

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

Derives

CsMessage

Version

1.0

readBoolean

Description

Reads a boolean from the stream message.

Declaration

```
bool readBoolean();
```

Returns

The Boolean value read

Throws

`CJMSException`

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

readByte

Description

Reads a byte value from the stream message.

Declaration

```
byte readByte();
```

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

Description

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

Declaration

```
int readBytes(byte[] value, int length);
```

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

Description

Reads a Unicode character value from the stream message.

Declaration

```
char readChar();
```

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

Description

Reads a double from the stream message.

Declaration

```
double readDouble();
```

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

Description

Reads a float from the stream message.

Declaration

```
float readFloat();
```

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

Description

Reads a 32-bit integer from the stream message.

Declaration

```
int readInt();
```

Returns

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

Throws

`CJMSException`

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

readLong

Description

Reads a 64-bit integer from the stream message.

Declaration

```
long readLong();
```

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

Description

Reads a 16-bit integer from the stream message.

Declaratio

```
short readShort();
```

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

Description

Reads a String from the stream message.

Declaration

```
string readString();
```

Returns

A string from the stream message

Throws

`CJMSException`

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

reset

Description

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

Declaration

```
void reset();
```

Throws

`CJMSException`

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

writeBoolean

Description

Writes a Boolean to the stream message.

Declaration

```
void writeBoolean(bool value);
```

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

Description

Writes a byte to the stream message.

Parameters

`value`

The byte value to be written.

Declaration

```
void writeByte(byte value);
```

Throws

`CJMSException`

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

writeBytes

Description

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(byte[] value, int length);
```

Parameters

`value`

The byte array value to be written length of value.

Throws

`CJMSException`

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

writeBytes

Description

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(byte[] value, int offset, int length);
```

Parameters

`value`

The byte array value to be written

`offset`

The initial offset within the byte array length.

Throws

`CJMSException`

If the JMS provider fails to write the message due to someinternal error.

writeChar

Description

Writes a char to the stream message.

Declaration

```
void writeChar(char value);
```

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

Description

Writes a double to the stream message.

Declaration

```
void writeDouble(double value);
```

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

Description

Writes a float to the stream message.

Declaration

```
void writeFloat(float value);
```

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

Description

Writes an int to the stream message.

Declaration

```
void writeInt(int value);
```

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

Description

Writes a long to the stream message.

Declaration

```
void writeLong(long value);
```

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

Description

Writes a short to the stream message.

Declaration

```
void writeShort(short value);
```

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

Description

Writes a string to the stream message.

Declaration

```
void writeString(string value);
```

Parameters

`value`

The String value to be written

Throws

`CJMSException`

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

CsTextMessage

A CsTextMessage object is used to send a message containing a string. It inherits from the CsMessage interface and adds a text message body.

Derives

`CsMessage`

Version

1.0

getText

Description

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

Declaration

```
string getText();
```

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

Description

Sets the string containing this message's data.

Declaration

```
void setText(string);
```

Parameters

`string`

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)

CsInitialContext

This class is the starting context for performing naming operations.

Version

1.0

Constructor

Description

Constructs an initial context using the supplied environment.

Version

1.0

Declaration

```
CsInitialContext(CsHashTable  env)
```

Parameters

`env`

Hashtable Contains the environment information.

Lookup

Description

Retrieves the named object.

Declaration

```
fobject Lookup(string adminObjectName);
```

Parameters

`adminObjectName`

Name of the Admin Object.

Returns

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

PTP

CsQueue

A CsQueue 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 CsDestination as a parameter, a CsQueue object is used as an argument.

Derives

`CsDestination`

Version

1.0

getQueueName

Description

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

Declaration

```
string getQueueName();
```

Returns

The queue name as a string

Throws

`CJMSException`

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

toString

Description

Returns a string representation of this object.

Declaration

```
string toString();
```

Returns

The provider-specific identity values for this queue.

CsQueueConnection

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

Derives

`CsConnection`

Version

1.0

createQueueSession

Description

Creates a QueueSession object.

Declaration

```
CsQueueSession createQueueSession(bool transacted,
int acknowledgeMode)
```

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

Description

Closes the connection. Since a provider typically allocates significant resources outside the JVM on behalf of a connection, clients should close these resources when theyare 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()
```

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

Description

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 bycalling the setClientID method.

Declaration

```
string getClientID()
```

Returns

The unique client identifier

Throws

`CJMSException`

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

setClientID

Description

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 CsConnectionFactory object and transparently assigned to the Connection object it creates.

Declaration

```
void setClientID(string clientID);
```

Parameters

`clientID`

The unique client identifier

Throws

`CJMSException`

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

start

Description

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();
```

Throws

`CJMSException`

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

stop

Description

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();
```

Throws

`CJMSException`

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

getExceptionListener

Description

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

Declaration

```
CsExceptionListener getExceptionListener();
```

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

Description

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(CsExceptionListener exceptionListener);
```

Parameters

`listener`

The exception listener

Throws

`CJMSException`

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

CsQueueConnectionFactory

A client uses a CsQueueConnectionFactory object to create CsQueueConnection objects with a point-to-point JMS provider.

Derives

`CsConnectionFactory`

Version

1.0

createQueueConnection

Description

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

```
CsQueueConnection createQueueConnection();
```

Returns

A newly created queue connection

Throws

`CJMSException`

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

createQueueConnection

Description

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

```
CsQueueConnection createQueueConnection(string username,
string password);
```

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.

CsQueueReceiver

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

Derives

`CsMessageConsumer`

Version

1.0

getQueue

Description

Gets the Queue associated with this queue receiver.

Declaration

```
CsQueue getQueue();
```

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

Description

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();
```

Throws

`CJMSException`

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

getMessageListener

Description

Gets the message consumer's MessageListener.

Declaration

```
CsMessageListener getMessageListener();
```

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

Description

Gets this message consumer's message selector expression.

Declaration

```
string getMessageSelector();
```

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 string).

Throws

`CJMSException`

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

receive

Description

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 is committed.

Declaration

```
CsMessage receive();
```

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

Description

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

```
CsMessage receive(long timeout);
```

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.

reveiveNoWait

Description

Receives the next message if one is immediately available.

Declaration

```
CsMessage receiveNoWait();
```

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.

set

Description

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(CsMessageListener messageListener)
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.

CsQueueRequestor

The CsQueueRequestor helper class simplifies making service requests.

Version

1.0

close

Description

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();
```

Throws

`CJMSException`

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

request

Description

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

```
CsMessage request(CsMessage msg);
```

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

Description

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

```
CsMessage request(CsMessage msg, long timeout);
```

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.

QueueBrowser

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

Version

1.0

close

Description

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();
```

Throws

`CJMSException`

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

getMessageSelector

Description

Gets this queue browser's message selector expression.

Declaration

```
string getMessageSelector();
```

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 string)

Throws

`CJMSException`

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

getQueue

Description

Gets the queue associated with this queue browser.

Declaration

```
CsQueue getQueue();
```

Returns

The queue

Throws

`CJMSException`

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

CsQueueSender

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

Derives

`CsMessageProducer`

Version

1.0

getQueue

Description

Gets the queue associated with this QueueSender.

Declaration

```
CsQueue getQueue();
```

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

Description

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();
```

Throws

`CJMSException`

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

getDeliveryMode

Description

Gets the producer's default delivery mode.

Declaration

```
int getDeliveryMode();
```

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

Description

Gets the destination associated with this MessageProducer.

Declaration

```
CsDestination getDestination();
```

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

Description

Gets an indication of whether message IDs are disabled.

Declaration

```
bool getDisableMessageID();
```

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

Description

Gets an indication of whether message timestamps are disabled

Declaration

```
bool getDisableMessageTimestamp();
```

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

Description

Gets the producer's default priority.

Declaration

```
int getPriority();
```

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

Description

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

Declaration

```
long getTimeToLive();
```

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

Description

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(CsDestination dest, CsMessage msg);
```

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

Description

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(CsDestination dest, CsMessage msg, int deliveryMode,
int priority, int timeToLive);
```

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

Description

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

Declaration

```
void send(CsMessage msg);
```

Parameters

`message`

The message to send

Throws

`CJMSException`

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

send

Description

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

Declaration

```
void send(CsMessage msg, int deliveryMode, int priority,
long timeToLive);
```

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

Description

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

Declaration

```
void setDeliveryMode(int deliveryMode);
```

Parameters

`deliveryMode`

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

Throws

`CJMSException`

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

setDisableMessageID

Description

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

Declaration

```
void setDisableMessageID(bool value);
```

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

Description

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

Declaration

```
void setDisableMessageTimestamp(bool value);
```

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

Description

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 expeditedpriority. Priority is set to 4 by default.

Declaration

```
void setPriority(int defaultPriority);
```

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

Description

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(long timeToLive);
```

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.

CsQueueSession

A CsQueueSession object provides methods for creating CsQueueReceiver, CsQueueSender,CsQueueBrowser, and CsTemporaryQueue objects.

Derives

Session

Version

1.0

close

Description

Closes the session.

Declaration

```
void close()
```

Throws

`CJMSException`

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

commit

Description

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

Declaration

```
void commit();
```

Throws

`CJMSException`

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

createBrowser

Description

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

Declaration

```
CsQueueBrowser createBrowser(CsQueue q);
```

Specified by

`createBrowser` in interface Session.

Parameters

`queue`

The Queue to access

Throws

`CJMSException`

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

createBrowser

Description

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

Declaration

```
CsQueueBrowser createBrowser(CsQueue q, string messageSelector);
```

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 string 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.

createBytesMesage

Description

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

Declaration

```
CsBytesMessage createBytesMesage();
```

Throws

`CJMSException`

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

createMapMessage

Description

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

```
CsMapMessage createMapMessage();
```

Throws

`CJMSException`

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

createQueue

Description

Creates a queue identity given a Queue name.

Declaration

```
CsQueue createQueue(string queueName);
```

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

Description

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

Declaration

```
CsStreamMessage createStreamMessage();
```

Throws

`CJMSException`

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

createTemporaryQueue

Description

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

Declaration

```
CsTemporaryQueue createTemporaryQueue();
```

Returns

A temporary queue identity

Throws

`CJMSException`

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

createTextMessage

Description

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

Declaration

```
CsTextMessage createTextMessage();
```

Returns

A temporary queue identity

Throws

`CJMSException`

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

createTextMessage

Description

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

Declaration

```
CsTextMessage createTextMessage(string text);
```

Parameters

`text`

The string used to initialize this message

Throws

`CJMSException`

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

getMessageListener

Description

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

Declaration

```
CsMessageListener getMessageListener()
```

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

Description

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

Declaration

```
void recover()
```

Throws

`CJMSException`

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

rollback

Description

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

Declaration

```
void rollback();
```

Throws

`CJMSException`

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

run

Description

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

Declaration

```
void run();
```

setMessageListener

Description

Sets the session's distinguished message listener.

Declaration

```
void setMessageListener(CsMessageListener messageListener);
```

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.

CsTemporaryQueue

A CsTemporaryQueue object is a unique CsQueue object created for the duration of a CsConnection. It is a system-defined queue that can be consumed only by the CsConnection that created it.

Derives

`CsDestination`

Version

1.0

remove

Description

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

Declaration

```
void remove();
```

Throws

`CJMSException`

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

Publish/Subscribe

CsTemporaryTopic

A CsTemporaryTopic object is a unique CsTopic object created for the duration of a CsConnection. It is a system-defined topic that can be consumed only by the CsConnection that created it.

Derives

`CsDestination`

Version

1.0

remove

Description

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

Declaration

```
void remove();
```

Throws

`CJMSException`

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

CsTopic

A CsTopic 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 CsDestination as a parameter, a CsTopic object may used as an argument.

Derives

`CsDestination`

Version

1.0

getTopicName

Description

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

Declaration

```
string getTopicName();
```

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

Description

Returns a string representation of this object.

Declaration

```
string toString();
```

Returns

The provider-specific identity values for this topic.

CsTopicConnection

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

Derives

`CsConnection`

Version

1.0

createTopicSession

Description

Creates a TopicSession object.

Declaration

```
CsTopicSession createTopicSession(bool transacted,
int acknowledgeMode);
```

Parameters

`transacted`

Indicates whether the session is transacted

`acknowledgeMode`

Indicates whether the consumer or the client will acknowledge any messages it receives.

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

Description

Closes the connection.

Declaration

```
void close();
```

Throws

`CJMSException`

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

getClientID

Description

Gets the client identifier for this connection.

Declaration

```
string getClientID();
```

Returns

The unique client identifier

Throws

`CJMSException`

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

setClientID

Description

Sets the client identifier for this connection.

Declaration/

```
void setClientID(string clientID);
```

Parameters

`clientID`

The unique client identifier

Throws

`CJMSException`

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

start

Description

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()
```

Throws

`CJMSException` -

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

stop

Description

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

Declaration

```
void stop()
```

Throws

`CJMSException`

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

getExceptionListener

Description

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

Declaration

```
CsExceptionListener getExceptionListener()
```

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

Description

Sets an exception listener for this connection.

Declaration

```
void setExceptionListener(CsExceptionListener exceptionListener);
```

Parameters

`listener`

The exception listener

Throws

`CJMSException`

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

CsTopicConnectionFactory

A client uses a CsTopicConnectionFactory object to create CsTopicConnection objects with a publish/subscribe JMS provider.

Derives

`CsConnectionFactory`

Version

1.0

createTopicConnection

Description

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

```
CsTopicConnection createTopicConnection();
```

Returns

A newly created topic connection

Throws

`CJMSException`

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

createTopicConnection

Description

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

```
CsTopicConnection createTopicConnection(string username, string
password);
```

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.

CsTopicPublisher

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

Derives

`CsMessageProducer`

Version

1.0

getTopic

Description

Gets the topic associated with this TopicPublisher.

Declaration

```
CsTopic getTopic();
```

Returns

This publisher's topic

Throws

`CJMSException`

If the JMS provider fails to get the topic for thisTopicPublisher due to some internal error.

publish

Description

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

Declaration

```
void publish(CsMessage msg);
```

Parameters

`message`

The message to publish

Throws

`CJMSException`

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

publish

Description

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

Declaration

```
void publish(CsMessage msg, int deliveryMode, int priority,
long timeToLive);
```

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

Description

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(CsTopic topic, CsMessage msg);
```

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

Description

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 requires the topic be supplied every time a message is published.

Declaration

```
void publish(CsTopic topic, CsMessage msg, int deliveryMode,
int priority, long timeToLive);
```

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 tosome internal err

close

Description

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();
```

Throws

`CJMSException`

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

getDeliveryMode

Description

Gets the producer's default delivery mode.

Declaration

```
int getDeliveryMode();
```

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

Description

Gets the destination associated with this MessageProducer.

Declaration

```
CsDestination getDestination();
```

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

Description

Gets an indication of whether message IDs are disabled.

Declaration

```
bool getDisableMessageID();
```

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

Description

Sets whether message timestamps are disabled.

Declaration

```
bool getDisableMesageTimestamp();
```

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

Description

Gets the producer's default priority.

Declaration

```
int getPriority();
```

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

Description

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

Declaration

```
long getTimeToLive();
```

Returns

The message time to live in milliseconds. zero is the unlimited time.

Throws

`CJMSException`

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

send

Description

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(CsDestination dest, CsMessage msg);
```

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

Description

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

Declaration

```
void send(CsDestination dest, CsMessage msg, int deliveryMode,
int priority, int timeToLive);
```

Parameters

`destination`

The destination to which the message has to be sent

`message`

The message to send

`deliveryMode`

The delivery mode to use

`priority`

The priority for this message

`timeToLive`

The lifetime of the message (in milliseconds)

Throws

`CJMSException`

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

send

Description

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

Declaration

```
void send(CsMessage msg);
```

Parameters

`message`

The message to send

Throws

`CJMSException`

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

send

Description

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

Declaration

```
void send(CsMessage msg, int deliveryMode, int priority,
mqlprong timeToLive);
```

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

Description

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

Declaration

```
void setDeliveryMode(int deliveryMode);
```

Parameters

`deliveryMode`

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

Throws

`CJMSException`

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

setDisableMessageID

Description

Sets whether message IDs are disabled.

Declaration

```
void setDisableMessageID(bool value);
```

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

Description

Sets whether message timestamps are disabled.

Declaration

```
void setDisableMessageTimestamp(bool value);
```

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

Description

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(int defaultPriority);
```

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

Description

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

Declaration

```
void setTimeToLive(long timeToLive);
```

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.

CsTopicRequestor

The CsTopicRequestor helper class simplifies making service requests.

Version

1.0

close

Declaration

```
void close();
```

Description

Closes the TopicRequestor and its session.

Throws

`CJMSException`

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

request

Description

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

```
CsMessage request(CsMessage msg);
```

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

Description

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

```
CsMessage request(CsMessage msg, long timeout);
```

Parameters

`message`

The message to be sent

`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.

CsTopicSession

A CsTopicSession object provides methods for creating CsTopicPublisher, CsTopicSubscriber, and CsTemporaryTopic objects. It also provides a method for deleting its client's durable subscribers.

Derives

`CsSession`

Version

1.0

createPublisher

Description

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 had sent previously.

Declaration

```
CsTopicPublisher createPublisher(CsTopic topic);
```

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 some internal error.

createSubscriber

Description

Creates a non-durable 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

```
CsTopicSubscriber createSubscriber(CsTopic topic);
```

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

Description

Creates a nondurable subscriber to the specified topic.

Declaration

```
CsTopicSubscriber createSubscriber(CsTopic topic,
string messageSelector, bool noLocal);
```

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 string indicates that there is no message selector for the message consumer.

`noLocal`

If set, inhibits the delivery of messages published by this connection.

Throws

`CJMSException`

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

close

Description

Closes the session.

Declaration

```
void close();
```

Throws

`CJMSException`

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

commit

Description

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

Declaration

```
void commit();
```

Throws

`CJMSException`

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

createBytesMessage

Description

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

Declaration

```
CsBytesMessage createBytesMesage()
```

Throws

`CJMSException`

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

createDurableSubscriber

Description

Creates a durable subscriber to the specified topic.

Declaration

```
CsTopicSubscriber createDurableSubscriber(CsTopic topic, string
name)
```

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

Description

Creates a durable subscriber to the specified topic.

Specified by

createDurableSubscriber in interface Session

Declaration

```
CsTopicSubscriber createDurableSubscriber(CsTopic topic, string
name,string messageSelector, bool noLocal);
```

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 string indicates that there is no message selector for the message consumer.

`noLocal`

If set, inhibits the delivery of messages published by this connection

Throws

`CJMSException`

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

createMapMessage

Description

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

```
CsMapMessage createMapMessage();
```

Throws

`CJMSException`

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

createStreamMessage

Description

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

Declaration

```
CsStreamMessage createStreamMessage();
```

Throws

`CJMSException`

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

createTemporaryQueue

Description

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

Declaration

```
CsTemporaryQueue createTemporaryQueue();
```

Returns

A temporary queue identity

Throws

`CJMSException`

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

createTextMessage

Description

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

Declaration

```
CsTextMessage createTextMessage();
```

Throws

`CJMSException`

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

createTextMessage

Description

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

Declaration

```
CsTextMessage createTextMessage(string text);
```

Parameters

`text`

The string used to initialize this message

Throws

`CJMSException`

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

createTopic

Description

Creates a topic identity given a Topic name.

Declaration

```
CsTopic createTopic(string topicName);
```

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

Description

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

Declaration

```
CsMessageListener getMessageListener();
```

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

Description

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

Declaration

```
void recover();
```

Throws

`CJMSException`

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

rollback

Description

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

Declaration

```
void rollback();
```

Throws

`CJMSException`

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

setMessageListener

Description

Sets the session's distinguished message listener.

Declaration

```
void setMessageListener(CsMessageListener messageListener);
```

Parameters

`messageListener`

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

Description

Unsubscribes a durable subscription that has been created by a client.

Declaration

```
void unsubscribe(string name);
```

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.

CsTopicSubscriber

A client uses a CsTopicSubscriber object to receive messages that have been published to a topic. A CsTopicSubscriber object is the publish/subscribe form of a message consumer.

Derives

`CsMessageConsumer`

Version

1.0

close

Description

Closes the message consumer.

Declaration

```
void close();
```

Throws

`CJMSException`

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

getMessageListener

Description

Gets the message consumer's MessageListener.

Declaration

```
CsMessageListener getMessageListener();
```

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

Description

Gets this message consumer's message selector expression.

Declaration

```
string getMessageSelector();
```

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 string).

Throws

`CJMSException`

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

receive

Description

Receives the next message produced for this message consumer. This callblocks 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

```
CsMessage receive();
```

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

Description

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

```
CsMessage receive(long timeout);
```

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

Description

Receives the next message if one is immediately available.

Declaration

```
CsMessage receiveNoWait();
```

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

Description

Sets the message consumer's MessageListener.

Declaration

```
void setMessageListener(CsMessageListener messageListener);
```

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

Description

Gets the NoLocal attribute for this subscriber. The default value for thisattribute is false.

Declaration

```
bool getNoLocal();
```

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

Description

Gets the Topic associated with this subscriber.

Declaration

```
CsTopic getTopic();
```

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.

Using the Samples Programs

5

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 sample files can be found in the csharp\samples folder in FioranoMQ installation folder. The samples programs illustrating the use of C#RTL for PubSub and PTP Operations are organized into two categories.

- **PubSub** This directory contains sample programs, which illustrate basic JMS, basic JMS HTTP and Request/Reply functionality.
- **PTP** This directory contains sample programs which illustrate basic JMS, basic JMS HTTP and Request/Reply mechanism.
- **VB.NET** This directory contains sample programs which illustrate JMS point-to-point and JMS Pub/Sub APIs.

Compiling and Running Samples

C#RTL need not be installed separately. The only required components are the C#DLL and the CppDLL. Reference these two components in the C#application project and you are ready to use the FMQC# RTL. Make sure to import FMQClient namespace in the application project.

Use the following batch files in scripts folder to compile and execute the sample files.

- **csclientbuild.bat** Use this batch file located in the scripts folder to compile any csharp application or sample.
- **csclientbuild.bat** Use this batch file to run the `.exe` which is created as output after compilation.

☞ For more details, refer to the readme.txt file in the samples folder.

Limitations of C#RTL

C#RTL provides a limited set of APIs for the end user, which just allows messaging using Pub/Sub or Sender/Receiver routines. There is no support for administration using the C# APIs. For this purpose the end user has to resort to the Java RTL only.

6

Frequently Asked Questions

Q1: What is the .NET framework?

The 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 as well as the agility to solve the challenges of deployment and operation of Internet-scale applications. The .NET Framework consists of three main parts: the common language runtime, a hierarchical set of unified class libraries, and a componentized version of Active Server Pages called ASP.NET.

Q2: Where do I get .Net?

To run an application with FMQ C# client libraraies, .Net framework needs to be installed first. This latest version can be downloaded from the Microsoft website.

Q3: What is FioranoMQ?

FioranoMQ (FMQ) is a communication platform that dramatically reduces the development time for network applications.

Q4: What is a FMQ client?

The FMQ 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.

Q5: What is the FMQ C#RTL

The FMQ C#RTL is the C# language version of the RTL implemented for the .NET platform. Using the FMQ C#RTL, .NET applications can now seamlessly communicate among themselves and with the whole range of applications written in any supported language on any supported platform.

Q6: What is FMQ.NET?

FMQ.NET is the client implementation of the messaging platform that provides for seamless connectivity among .NET applications and any other java, C, C++ application on any supported platform.

Q7: What is GC?

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.

Q8: How do I solve the error "Null Reference Exception"?

This exception occurs when a JMS object created by the user was garbage collected when still in use. This can happen when the only reference to the managed object is with the unmanaged code. This situation can be avoided by calling the GC.KeepAlive on the JMS objects until where they may be needed.

Q9: How to run Csharp assembly in HTTP/HTTPS/SSL mode?

The C # RTL supports two protocols - TCP and HTTPS.

The TCP version only supports TCP connections to the JMS server.

The HTTPS version supports HTTP, HTTPS and SSL connections with the server. The *csclientbuild.bat* script, available with the installation package, has the provision to build the C # RTL with HTTPS assemblies. Please read the script for instructions.

Also, ensure that the following libraries are in system path while running the application with HTTPS assemblies.

1. pthreadVC.dll
2. gnu_regex.dll
3. libeay32.dll
4. ssleay32.dll
5. zlib.dll

These libraries can be found with the FMQ client installation package.