



Copyright (c) 2008-2010, Fiorano Software Pte. Ltd. and affiliates

All rights reserved.

This software is the confidential and proprietary information of Fiorano Software ("Confidential Information"). You shall not disclose such ("Confidential Information") and shall use it only in accordance with the terms of the license agreement enclosed with this product or entered into with Fiorano.

**Fiorano**

# Content

---

<b>Chapter 1: Introduction</b> .....	<b>18</b>
<b>Chapter 2: Datatypes and Constants</b> .....	<b>19</b>
Basic Data Types and their Sizes .....	19
C++RTL Constants.....	19
Naming convention .....	20
<b>Chapter 3: Error Handling</b> .....	<b>21</b>
<b>Chapter 4: The DotNet Version</b> .....	<b>22</b>
Installation .....	22
Using DotNet samples .....	22
Organization of samples .....	22
Compiling the samples .....	23
Running the samples .....	23
<b>Chapter 5: API Reference</b> .....	<b>24</b>
Helpers .....	24
CHashTable .....	24
Constructor.....	24
Put .....	24
Get .....	24
RemoveElement.....	25
HashTableEnumerator .....	25
hasMoreElemets .....	25
nextKeyElement .....	25
nextValueElement .....	25
JMS Interfaces.....	26
CAdvisoryMessage .....	26
getAdvisoryMsgString .....	26
getAMState .....	26
isActive .....	26
isDisconnected .....	27
isRevalidating .....	27
isFailed.....	27
isTransferring .....	28
isTransferComplete .....	28

CAdvisoryMsgListener .....	29
onAdvisoryMessage .....	29
CByteMessage .....	29
getBodyLength .....	29
readBoolean .....	30
readByte .....	30
readBytes .....	30
readChar .....	31
readDouble .....	31
readFloat .....	32
readInt .....	32
readLong .....	32
readShort .....	33
readUnsignedByte .....	33
readUnsignedShort .....	33
readUTF .....	34
reset .....	34
writeBoolean .....	34
writeByte .....	35
writeBytes .....	35
writeChar .....	36
writeDouble .....	36
writeFloat .....	36
writeInt .....	37
writeLong .....	37
writeShort .....	38
writeUTF .....	38
CConnection .....	38
CConnectionConsumer .....	39
CConnectionFactory .....	39
CDestination .....	39
CFioranoException .....	39
checkForException .....	39
CJMSException .....	39
Constructor .....	39
Constructor .....	40
checkForException .....	40
Another over-loaded function .....	40
printStackTrace .....	41
getStackTrace .....	41
getErrorCode .....	41
getLinkedException .....	41
CExceptionListener .....	41
onException .....	42
CMapMessage .....	42
getBoolean .....	42
getByte .....	43
getChar .....	43

getDouble	43
getFloat	44
getInt	44
getLong	45
getMapNames	45
getMapNamesHTEnum	46
getShort	46
getString	47
itemExists	47
setBoolean	47
setByte	48
setBytes	48
setBytes	49
setChar	49
setDouble	50
setFloat	50
setInt	51
setLong	51
setShort	52
setString	52
CMessage	52
Acknowledge	53
ClearBody	53
clearProperties	53
getBooleanProperty	54
getBytesProperty	54
.getDoubleProperty	55
.getFloatProperty	55
.getIntProperty	55
getJMSCorrelationID	56
getJMSCorrelationIDAsBytes	56
getJMSDeliveryMode	57
getJMSDestination	57
getJMSExpiration	58
getJMSMessageID	58
getJMSPriority	59
getJMSRedelivered	59
getJMSReplyTo	59
getJMSTimestamp	60
getJMSType	60
getLongProperty	60
getObjectProperty	61
getPropertyNames	61
getShortProperty	62
getStringProperty	62
getStringProperty_unicode	63
getMessageType	63
propertyExists	63

setBooleanProperty .....	64
setByteProperty .....	64
setDoubleProperty .....	65
setFloatProperty .....	65
setIntProperty .....	65
setJMSCorrelationID .....	66
setJMSDeliveryMode .....	66
setJMSDestination .....	67
setJMSExpiration .....	67
setJMSMessageID .....	67
setJMSPriority .....	68
setJMSRedelivered .....	68
setJMSReplyTo .....	69
setJMSTimestamp .....	69
setJMSType .....	70
setLongProperty .....	70
setObjectProperty .....	70
setshortProperty .....	71
setStringProperty .....	71
CMessageConsumer .....	72
CMessageListener .....	72
CServerSession .....	72
CServerSessionPool .....	72
CSession .....	72
CStreamMessage .....	73
readBoolean .....	73
readByte .....	73
readBytes .....	74
readChar .....	74
readDouble .....	75
readFloat .....	75
readInt .....	75
readLong .....	76
readShort .....	76
readString .....	76
reset .....	77
writeBoolean .....	77
writeByte .....	77
writeBytes .....	78
writeBytes .....	78
writeChar .....	79
writeDouble .....	79
writeFloat .....	80
writeInt .....	80
writeLong .....	80
writeShort .....	81
writeString .....	81
CTextMessage .....	81

getText .....	82
setText .....	82
Naming and Lookup (JNDI) .....	82
CInitialContext .....	82
Constructor .....	83
Lookup .....	83
LookupQCF .....	83
LookupTCF .....	84
PTP .....	84
CQueue .....	84
getQueueName .....	84
toString .....	85
CQueueConnection .....	85
createQueueSession .....	85
close .....	86
getClientID .....	86
setAdvisoryMessageListener .....	86
setClientID .....	87
start .....	87
stop .....	88
getExceptionListener .....	88
setExceptionListener .....	88
CQueueConnectionFactory .....	89
createQueueConnection .....	89
createQueueConnection .....	89
CQueueReceiver .....	90
getQueue .....	90
close .....	90
getMessageListener .....	91
getMessageSelector .....	91
receive .....	91
recieve .....	92
receiveNoWait .....	92
setMessageListener .....	93
CQueueRequestor .....	93
close .....	93
request .....	93
request .....	94
CQueueBrowser .....	94
close .....	95
getMessageSelector .....	95
getQueue .....	95
CQueueSender .....	96
getQueue .....	96
close .....	96
getDeliveryMode .....	96
getDestination .....	97
getDisableMessageID .....	97

getDisableMessageTimestamp.....	97
getPriority.....	98
getTimeToLive.....	98
send.....	99
send.....	99
send.....	100
send.....	100
setDeliveryMode.....	101
setDisableMessageID.....	101
setDisableMessageTimeStamp.....	101
setPriority.....	102
setTimeToLive.....	102
CQueueSession.....	103
close.....	103
commit.....	103
createBrowser.....	103
createBrowser.....	104
createReceiver.....	104
createReceiver.....	105
createSender.....	105
createBytesMessage.....	106
createMapMessage.....	106
createQueue.....	107
createStreamMessage.....	107
createTemporaryQueue.....	107
createTextMessage.....	108
createTextMessage.....	108
getMessageListener.....	108
recover.....	109
rollback.....	109
run.....	109
setMessageListener.....	110
CTemporaryQueue.....	110
remove.....	110
Publish/Subscribe.....	110
CTemporaryTopic.....	110
remove.....	111
CTopic.....	111
getTopicName.....	111
toString.....	111
CTopicConnection.....	112
createTopicSession.....	112
close.....	113
getClientID.....	113
setAdvisoryMessageListener.....	113
setClientID.....	114
start.....	114
stop.....	114



getExceptionListener .....	115
setExceptionListener.....	115
CTopicConnectionFactory .....	115
createTopicConnection.....	115
createTopicConnection .....	116
CTopicPublisher.....	116
getTopic .....	117
publish .....	117
publish .....	117
publish .....	118
publish .....	118
close .....	119
getDeliveryMode .....	119
getDestination .....	120
getDisableMessageID .....	120
getDisableMessageTimestamp .....	120
getPriority.....	121
getTimeToLive .....	121
send .....	121
send .....	122
send .....	123
send .....	123
setDeliveryMode .....	123
setDisableMessageID.....	124
setDisableMessageTimestamp .....	124
setPriority.....	125
setTimeToLive .....	125
CTopicRequestor .....	125
close .....	126
request .....	126
request .....	126
CTopicSession.....	127
createPublisher .....	127
createSubscriber .....	127
createSubscriber .....	128
close .....	128
commit.....	129
createBytesMessage .....	129
createDurableSubscriber .....	129
createDurableSubscriber .....	130
createMapMessage .....	130
createStreamMessage.....	131
createTemporaryQueue.....	131
createTextMessage.....	131
createTextMessage.....	132
createTopic .....	132
getMessageListener.....	132
recover.....	133

rollback .....	133
setMessageListener .....	133
unsubscribe .....	134
CTopicSubscriber .....	134
close .....	134
getMessageListener .....	135
getMessageSelector .....	135
receive .....	135
receive .....	136
receiveNoWait .....	136
setMessageListener .....	137
getNoLocal .....	137
getTopic .....	137
CLogHandler .....	138
setLoggerName .....	138
getLogHandler .....	138
setTraceLevel .....	138
logData .....	138
CCSPManager .....	139
Constructor .....	139
createCSPBrowser .....	139
CCSPBrowser .....	139
getAllConnections .....	139
getTopicsForConnection .....	139
getQueuesForConnection .....	140
browseMessagesOnQueue .....	140
browseMessagesOnQueue .....	140
browseMessagesOnTopic .....	140
browseMessagesOnTopic .....	140
numberOfMessagesInQueue .....	141
numberOfMessagesInTopic .....	141
CCSPEnumeration .....	141
nextElement .....	141
Large Message Support .....	142
CfioranoConnection .....	142
getUnfinishedMessagesToSend .....	142
getUnfinishedMessagesToReceive .....	142
CRecoverableMessagesEnum .....	142
hasMoreElements .....	142
nextElement .....	142
CLargeMessage .....	142
getMessageStatus .....	142
setLMStatusListener .....	143
getLMStatusListener .....	143
saveTo .....	143
resumeSaveTo .....	143
resumeSend .....	143
cancelAllTransfers .....	143

cancelTransfer .....	143
suspendAllTransfers .....	144
suspendTransfer .....	144
setFragmentSize .....	144
getFragmentSize .....	144
setWindowSize .....	144
getWindowSize .....	144
setRequestTimeoutInterval .....	144
getRequestTimeoutInterval .....	145
setResponseTimeoutInterval .....	145
getResponseTimeoutInterval .....	145
CLMStatusListener .....	145
onLMStatus .....	145
CLMTransferStatus .....	145
getBytesTransferred .....	145
getBytesToTransfer .....	145
getLastFragmentID .....	146
getPercentageProgress .....	146
getStatus .....	146
isTransferComplete .....	146
isTransferCompleteForAll .....	146
getLargeMessage .....	147
getConsumerID .....	147
Administration API .....	147
CAdminConnectionFactory .....	147
createAdminConnectionDefParams .....	147
createAdminConnection .....	147
CAdminConnection .....	148
getMQAdminService .....	148
Close .....	148
CMQAdminService .....	149
getNumberOfActiveClientConnections .....	149
getDurableSubscribersForTopic .....	149
getClientIDs .....	150
getPTPClientIDs .....	150
getPubSubClientIDs .....	150
getSubscriberIDs .....	151
getSubscriptionTopicName .....	151
getNumberOfDeliverableMessages1 .....	152
getNumberOfDeliverableMessages2 .....	152
getNumberOfUndeletedMessages .....	153
Unsubscribe .....	153
purgeSubscriptionMessages .....	154
createTopic .....	154
createQueue .....	155
deleteTopic .....	155
deleteQueue .....	155
createTopicConnectionFactory .....	156

createQueueConnectionFactory .....	156
deleteQueueConnectionFactory .....	157
deleteTopicConnectionFactory .....	157
getCurrentUsers .....	157
restartServer .....	158
shutdownServer .....	158
shutDownActiveHAServer .....	159
shutDownPassiveHAServer .....	159
purgeQueueMessages1 .....	159
purgeQueueMessages2 .....	160
showStatusOfAllQueues .....	160
deleteMessagesOnServer .....	161
loadAdminObjects .....	161
CTopicMetaData .....	162
setName .....	162
setDescription .....	162
getName .....	163
getDescription .....	163
CQueueMetaData .....	163
setName .....	163
setDescription .....	164
getName .....	164
getDescription .....	165
CTopicConnectionFactoryMetaData .....	165
allowAutoRevalidation .....	165
allowDurableConnections .....	166
areDurableConnectionsAllowed .....	166
compareConnectURLs .....	166
disableCSPStoredMessageSend .....	167
disablePing .....	167
disableReaderCache .....	168
getAutoDispatch .....	168
getBatchTimeoutmqinterval .....	168
getCreateLocalSocket .....	169
getAdminConnectionReconnectmqinterval .....	169
getDurableConnectionReconnectmqinterval .....	170
getClientProxyURL .....	170
getCompressionManager .....	170
getConnectionClientID .....	171
getConnectURL .....	171
getCSPUpdateFrequency .....	171
getDurableConnectionBaseDir .....	171
getFactoryMetadataParams .....	172
getHTTPProxyURL .....	172
getLazyRSCreation .....	172
getLMSEnabled .....	173
getLookUpPreferredServer .....	173
getMaxAdminConnectionReconnectAttempts .....	174

getMaxDurableConnectionReconnectAttempts .....	174
getMaxSocketCreationTries .....	174
getName .....	175
getPort .....	175
getProxyCredentials .....	176
getProxyPrincipal .....	176
getProxyType .....	176
getPublishBehaviourInAutoRevalidation .....	177
getPublishWaitDuringCSPSyncp .....	177
getSecondaryConnectURL .....	177
getSecurityProtocol .....	178
getServerProxyURL .....	178
getSleepSocketCreationTries .....	178
getSocketTimeout .....	179
getSOCKSProxyURL .....	179
getStateTransitionOnReceiveSocket .....	180
getTCPBatchSize .....	180
getTransportProtocol .....	180
isAutoRevalidationEnabled .....	181
isBatchingEnabled .....	181
isConnectURLUpdationAllowed .....	181
isCSPStoredMessageSendDisabled .....	182
isPingDisabled .....	182
isReaderCacheDisabled .....	183
isSingleSocketForSendReceiveEnabled .....	183
isSocketKeepAliveEnabled .....	183
setAdminConnectionReconnectmqinterval .....	184
setAutoDispatch .....	184
setBatchTimeoutmqinterval .....	185
setCreateLocalSocket .....	185
setDurableConnectionReconnectmqinterval .....	186
setSecondaryConnectURL .....	186
setBatchingEnabled .....	187
setClientProxyURL .....	187
setCompressionManager .....	188
setConnectionClientID .....	188
setConnectURL .....	189
setCSPUpdateFrequency .....	189
setDurableConnectionBaseDir .....	190
setFactoryMetadataParams .....	190
setHTTPProxyURL .....	191
setIsForLPC .....	191
setLazyRSCreation .....	192
setLMSEnabled .....	192
setLookUpPreferredServer .....	192
setMaxAdminConnectionReconnectAttempts .....	193
setMaxDurableConnectionReconnectAttempts .....	193
setMaxSocketCreationTries .....	194

setName .....	194
setProxyCredentials .....	195
setProxyPrincipal .....	195
setProxyType .....	196
setPublishBehaviourInAutoRevalidation .....	196
setPublishWaitDuringCSPSyncp .....	197
setSecondaryConnectURL .....	197
setSecurityProtocol .....	198
setServerProxyURL .....	198
setSleepSocketCreationTries .....	199
setSocketKeepAlive .....	199
setSocketTimeout .....	200
setSOCKSProxyURL .....	200
setStateTransitionOnReceiveSocket .....	201
setTCPBatchSize .....	201
setTransportProtocol .....	202
setUseSingleSocketForSendReceive .....	202
updateConnectURL .....	203
CQueueConnectionFactoryMetaData .....	203
allowAutoRevalidation .....	204
allowDurableConnections .....	204
areDurableConnectionsAllowed .....	204
compareConnectURLs .....	205
disableCSPStoredMessageSend .....	205
disablePing .....	206
disableReaderCache .....	206
getAdminConnectionReconnectmqinterval .....	207
getAutoDispatch .....	207
getBatchTimeoutmqinterval .....	207
getClientProxyURL .....	208
getCompressionManager .....	208
getConnectionClientID .....	208
getConnectURL .....	209
getCreateLocalSocket .....	209
getCSPUpdateFrequency .....	210
getDurableConnectionBaseDir .....	210
getDurableConnectionReconnectmqinterval .....	210
getFactoryMetadataParams .....	211
getHTTPProxyURL .....	211
getLazyRSCreation .....	211
getLMSEnabled .....	212
getLookUpPreferredServer .....	212
getMaxAdminConnectionReconnectAttempts .....	213
getMaxDurableConnectionReconnectAttempts .....	213
getMaxSocketCreationTries .....	213
getName .....	214
getPort .....	214
getProxyCredentials .....	214

getProxyPrincipal .....	215
getProxyType .....	215
getPublishBehaviourInAutoRevalidation .....	216
getPublishWaitDuringCSPSyncp .....	216
getSecondaryConnectURL .....	216
getSecurityProtocol .....	217
getServerProxyURL .....	217
getSleepSocketCreationTries .....	217
getSocketTimeout .....	218
getSOCKSProxyURL .....	218
getStateTransitionOnReceiveSocket .....	219
getTCPBatchSize .....	219
getTransportProtocol .....	219
isAutoRevalidationEnabled .....	220
isBatchingEnabled .....	220
isConnectURLUpdationAllowed .....	220
isCSPStoredMessageSendDisabled .....	221
isForLPC .....	221
isReaderCacheDisabled .....	222
isSingleSocketForSendReceiveEnabled .....	222
isSocketKeepAliveEnabled .....	222
setAdminConnectionReconnectmqinterval .....	223
setAutoDispatch .....	223
setBatchingEnabled .....	224
setClientProxyURL .....	224
setCompressionManager .....	225
setConnectionClientID .....	225
setConnectURL .....	226
setCreateLocalSocket .....	226
setCSPUpdateFrequency .....	227
setDurableConnectionBaseDir .....	227
setDurableConnectionReconnectmqinterval .....	228
setFactoryMetadataParams .....	228
setHTTPProxyURL .....	229
setIsForLPC .....	229
setLazyRSCreation .....	230
setLMSEnabled .....	230
setLookUpPreferredServer .....	231
setMaxAdminConnectionReconnectAttempts .....	231
setMaxDurableConnectionReconnectAttempts .....	232
setMaxSocketCreationTries .....	232
setName .....	233
setProxyCredentials .....	233
setProxyPrincipal .....	234
setProxyType .....	234
setPublishBehaviourInAutoRevalidation .....	235
setPublishWaitDuringCSPSyncp .....	235
setSecondaryConnectURL .....	236

setSecurityProtocol .....	236
setServerProxyURL.....	237
setSleepSocketCreationTries.....	237
setSocketKeepAlive .....	238
setSocketTimeout .....	238
setSOCKSProxyURL.....	239
setStateTransitionOnReceiveSocket .....	239
setTCPBatchSize .....	240
setTransportProtocol .....	240
setUseSingleSocketForSendReceive .....	241
updateConnectURL.....	241

## Chapter 6: Using Sample Programs ..... 242

Organization of Samples Provided .....	242
Compiling and Running the Samples .....	242
Limitations of C++RTL.....	242

## Chapter 7: Native C++ Runtime Examples ..... 243

PTP Samples .....	244
Admin.....	244
Basic .....	244
Browser .....	244
HTTP .....	245
HTTPS .....	245
MsgSel.....	246
Mtptp .....	246
reqrep .....	246
basic.....	246
TimedOut .....	247
SSL.....	247
Transaction .....	248
PubSub Samples.....	248
Admin.....	248
Basic .....	248
Dursub .....	249
HTTP .....	250
HTTPS .....	250
Msgsel .....	251
Mtpubsub .....	251
Reqrep.....	252
Basic.....	252
TimedOut .....	252
SSL.....	253
Transaction .....	253



Chapter 8: Frequently Asked Questions..... 254

# Chapter 1: Introduction

---

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

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

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

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

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

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

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

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

# Chapter 2: Datatypes and Constants

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

## Basic Data Types and their Sizes

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

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

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

## C++RTL Constants

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

Class	Constant
<code>Cmessage</code>	<pre>static const int CDEFAULT_DELIVERY_MODE static const int CDEFAULT_PRIORITY static const int CDEFAULT_TIME_TO_LIVE</pre>
<code>CdeliveryMode</code>	<pre>static const int CNON_PERSISTENT; static const int CPERSISTENT SET_CPERSISTENT;</pre>

Class	Constant
CJMSSession	static const int CAUTO_ACKNOWLEDGE static const int CCLIENT_ACKNOWLEDGE static const int CDUPS_OK_ACKNOWLEDGE static const int CSESSION_TRANSACTED

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

## Naming convention

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

Type	Naming Convention	Example
Class	C<JMSSession name>	CTopicPublisher(JMS:TopicPublisher)
Constant	C<JMS constants name>	CNON_PERSISTENT()
Function	same as in JMS spec	publish

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

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

## Chapter 3: Error Handling

---

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

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

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

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

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

# Chapter 4: The DotNet Version

---

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

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

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

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

## Installation

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

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

## Using DotNet samples

### Organization of samples

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

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

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

## Compiling the samples

To run the samples using FioranoMQ,

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

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

## Running the samples

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

# Chapter 5: API Reference

---

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

The organization of this document is summarized as follows.

## Helpers

### CHashTable

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

#### Constructor

Constructs a new, empty hashtable.

#### Declaration

```
CHashTable()  
    throw (CJMSEException *);
```

#### Put

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

#### Declaration

```
mqobject Put(mqcstring key, mqobject value)  
    throw (CJMSEException *);
```

#### Get

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

#### Declaration

```
mqobject Get(mqcstring key) FMQCONST  
    throw (CJMSEException *);
```



## RemoveElement

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

### Declaration

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

## HashTableEnumerator

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

### hasMoreElemets

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

### Declaration

```
mqboolean hasMoreElements();
```

### nextKeyElement

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

### Declaration

```
mqcstring nextKeyElement();
```

### Returns

Returns the next Key value

### nextValueElement

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

### Declaration

```
mqobject nextValueElement();
```

### Returns

Returns the next value Element

## JMS Interfaces

### CAdvisoryMessage

Base class for all advisory messages.

#### getAdvisoryMsgString

Returns a Unicode string for the advisory message.

##### Declaration

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

##### Returns

The Unicode string value for the advisory message string

##### Throws

CJMSException

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

#### getAMState

Returns a mqint object representing the Advisory message state

##### Declaration

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

##### Returns

The advisory message state as mqint object

##### Throws

CJMSException

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

#### isActive

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

##### Declaration

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

### Returns

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

### Throws

CJMSException

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

### isDisconnected

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

### Declaration

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

### Returns

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

### Throws

CJMSException

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

### isRevalidating

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

### Declaration

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

### Returns

Returns TRUE if revalidating thread is active, FALSE otherwise.

### Throws

CJMSException

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

### isFailed

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

### Declaration

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

### Returns

Returns TRUE if is failed, FALSE otherwise.

### Throws

CJMSException

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

### isTransferring

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

### Declaration

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

### Returns

Returns TRUE if is transferring, FALSE otherwise.

### Throws

CJMSException

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

### isTransferComplete

Returns mqboolean value representing the transfer status.

### Declaration

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

### Returns

Returns TRUE if the transfer is complete, FALSE otherwise.

### Throws

CJMSException

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

## CAdvisoryMsgListener

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

### onAdvisoryMessage

Passes a message to the listener.

#### Declaration

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

#### Parameters

msg

The message passed to the listener

#### Throws

CJMSException

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

## CByteMessage

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

#### Derives

CMessage

### getBodyLength

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

#### Declaration

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

#### Returns

Number of bytes in the message.

#### Throws

CJMSException

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

### readBoolean

Reads a boolean from the bytes message stream.

#### Declaration

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

#### Returns

The boolean value read

#### Throws

CJMSEException

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

### readByte

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

#### Declaration

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

#### Returns

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

#### Throws

CJMSEException

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

### readBytes

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

#### Declaration

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

## Parameters

value

The buffer into which the data is read.

Length

The length of the buffer.

## Returns

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

## Throws

CJMSException

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

## readChar

Reads a Unicode character value from the bytes message stream.

## Declaration

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

## Returns

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

## Throws

CJMSException

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

## readDouble

Reads a double from the bytes message stream.

## Declaration

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

## Returns

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

## Throws

CJMSException

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

### readFloat

Reads a float from the bytes message stream.

#### Declaration

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

#### Returns

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

#### Throws

CJMSException

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

### readInt

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

#### Declaration

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

#### Returns

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

#### Throws

CJMSException

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

### readLong

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

#### Declaration

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

#### Returns

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



### Throws

CJMSException

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

### readShort

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

### Declaration

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

### Returns

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

### Throws

CJMSException

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

### readUnsignedByte

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

### Declaration

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

### Returns

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

### Throws

CJMSException

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

### readUnsignedShort

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

### Declaration

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

## Returns

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

## Throws

CJMSEException

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

## readUTF

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

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

## Declaration

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

## Returns

A Unicode mqcstring from the bytes message stream

## Throws

CJMSEException

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

## reset

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

## Declaration

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

## Throws

CJMSEException

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

## writeBoolean

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

### Declaration

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

### Parameters

value

The boolean value to be written

### Throws

CJMSException

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

### writeByte

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

### Declaration

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

### Parameters:

value

The byte value to be written

### Throws

CJMSException

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

### writeBytes

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

### Declaration

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

### Parameters

value

The byte array value to be written  
offset

The initial offset within the byte array  
length

The number of bytes to use

### Throws

CJMSException

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

### writeChar

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

### Declaration

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

### Parameters

value

The char value to be written

### Throws

CJMSException

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

### writeDouble

Reads a double from the bytes message stream.

### Declaration

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

### Returns

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

### Throws

CJMSException

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

### writeFloat

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

### Declaration

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

### Parameters

value

The float value to be written

### Throws

CJMSException

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

### writeInt

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

### Declaration

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

### Parameters

value

The int to be written

### Throws

CJMSException

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

### writeLong

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

### Declaration

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

### Parameters

value

The long to be written

### Throws

CJMSException

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

### writeShort

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

#### Declaration

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

#### Parameters

value

The short to be written

#### Throws

CJMSEException

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

### writeUTF

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

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

#### Declaration

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

#### Parameters

value

The String value to be written

#### Throws

CJMSEException

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

### CConnection

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

**Base for**

CQueueConnection CTopicConnection

## CConnectionConsumer

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

## CConnectionFactory

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

**Base for**

CTopicConnectionFactory CQueueConnectionFactory

## CDestination

A CDestination object encapsulates a provider-specific address.

**Base for**

Ctopic CQueue CTemporaryTopic CTemporaryQueue

## CFioranoException

This is the root class of all exceptions.

public

### checkForException

Utility Function that throws CJMSEException if an exception had occurred.

**Declaration**

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

## CJMSEException

### Constructor

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

### Declaration

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

### Parameters

errCode

The error code that identifies the error

### Constructor

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

### Declaration

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

### Parameters

errCode

The error code that identifies the error

errDesc

description of the error

### checkForException

Utility Function that throws CJMSException if an exception had occurred.

### Declaration

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

### Parameters

errCode

The error code that identifies the error

### Another over-loaded function

Utility Function that throws CJMSException if an exception had occurred.

### Declaration

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

### Parameters

errCode



The error code that identifies the error  
errDesc

of the error

### printStackTrace

Prints the Function stack trace on the console

#### Declaration

```
void printStackTrace();
```

### getStackTrace

Gets the stack trace of the function

#### Declaration

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

### getErrorCode

Gets the vendor-specific error code.

#### Declaration

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

#### Returns

The vendor-specific error code

### getLinkedException

Gets the exception linked to this one

#### Declaration

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

#### Returns

The linked exception

## CExceptionListener

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

## onException

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

### Declaration

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

### Parameters

msg

Message handle

## CMapMessage

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

### Derives

CMessage

## getBoolean

Returns the boolean value with the specified name.

### Declaration

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

### Parameters

name

The name of the boolean

### Returns

The boolean value with the specified name

### Throws

CJMSEException

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

## getBytes

Returns the byte value with the specified name.

### Declaration

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

### Parameters

name

The name of the byte

### Returns

The byte value with the specified name.

### Throws

CJMSEException

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

## getChar

Returns the Unicode character value with the specified name.

### Declaration

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

### Parameters

name

The name of the Unicode character

### Returns

The Unicode character value with the specified name

### Throws

CJMSEException

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

## getDouble

Returns the double value with the specified name.

### Declaration

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

### Parameters

name

The name of the double

### Returns

The double value with the specified name

### Throws

CJMSEException

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

## getFloat

Returns the float value with the specified name.

### Declaration

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

### Parameters

name

The name of the float

### Returns

The float value with the specified name

### Throws

CJMSEException

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

## getInt

Returns the int value with the specified name.

### Declaration

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

### Parameters

name

The name of the int

### Returns

The int value with the specified name

### Throws

CJMSException

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

### getLong

Returns the long value with the specified name.

### Declaration

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

### Parameters

name

The name of the long

### Returns

The long value with the specified name

### Throws

CJMSException

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

### getMapNames

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

### Declaration

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

### Returns

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

### Throws

CJMSException

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

### getMapNamesHTEnum

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

### Declaration

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

### Returns

An enumeration of all the names in this MapMessage

### Throws

CJMSException

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

### getShort

Returns the short value with the specified name.

### Declaration

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

### Parameters

name

The name of the short

### Returns

The short value with the specified name

### Throws

CJMSException

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

## getString

Returns the String value with the specified name.

### Declaration

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

### Parameters

name

The name of the String

### Returns

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

### Throws

CJMSException

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

## itemExists

Indicates whether an item exists in this MapMessage object.

### Declaration

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

### Parameters

name

The name of the item to test

### Returns

TRUE if the item exists

### Throws

CJMSException

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

## setBoolean

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

### Declaration

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

### Parameters

name

The name of the boolean

value

The boolean value to set in the Map

### Throws

CJMSEException

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

## setByte

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

### Declaration

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

### Parameters

name

The name of the byte

value

The byte value to set in the Map

### Throws

CJMSEException

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

## setBytes

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

### Declaration

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



## 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

CJMSEException

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

## setBytes

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

## Declaration

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

## 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

CJMSEException

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

## setChar

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

## Declaration

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

## Parameters

name

The name of the Unicode character value

The Unicode character value to set in the Map

## Throws

MSException

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

## setDouble

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

## Declaration

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

## Parameters

name

The name of the double value

The double value to set in the Map

## Throws

CJMSException

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

## setFloat

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

## Declaration

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

## Parameters

name

The name of the float value

The float value to set in the Map

### Throws

CJMSException

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

### setInt

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

### Declaration

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

### Parameters

name

The name of the int  
value

The int value to set in the Map

### Throws

CJMSException

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

### setLong

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

### Declaration

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

### Parameters

name

The name of the long  
value

The long value to set in the Map

### Throws

CJMSException

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

## setShort

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

### Declaration

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

### Parameters

name

The name of the short  
value

The short value to set in the Map

### Throws

CJMSEException

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

## setString

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

### Declaration

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

### Parameters

name

The name of the String  
value

The String value to set in the Map

### Throws

CJMSEException

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

## CMessage

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

### Base for

CTextMessage CBytesMessage CMapMessage CObjectMessage

and

CStreamMessage

## Acknowledge

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

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

### Declaration

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

### Throws

CJMSEException

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

## ClearBody

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

### Declaration

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

### Throws

CJMSEException

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

## clearProperties

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

### Declaration

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

### Throws

CJMSException

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

### getBooleanProperty

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

### Declaration

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

### Parameters

name

The name of the boolean property

### Returns

The boolean property value for the specified name

### Throws

CJMSException

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

### getBytesProperty

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

### Declaration

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

### Parameters

name

The name of the byte property

### Returns

The byte property value for the specified name

### Throws

CJMSException

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

## getDoubleProperty

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

### Declaration

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

### 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

CJMSEException

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

## getFloatProperty

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

### Declaration

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

### Parameters

name

The name of the double property

### Returns

The double property value for the specified name

### Throws

CJMSEException

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

## getIntProperty

Description

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

### Declaration

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

### Parameters

name

The name of the int property

### Returns

The int property value for the specified name

### Throws

CJMSEException

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

## getJMSCorrelationID

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

### Declaration

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

### Returns

The correlation ID of a message as a String

### Throws

CJMSEException

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

## getJMSCorrelationIDsAsBytes

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

### Returns

The correlation ID of a message as an array of bytes

### Throws

JMSEException

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



### Declaration

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

### getJMSDeliveryMode

Gets the DeliveryMode value specified for this message.

### Declaration

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

### Returns

The delivery mode for this message

### Throws

CJMSException

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

### getJMSDestination

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

### Declaration

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

### Returns

The destination of this message

### Throws

CJMSException

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

## getJMSExpiration

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

### Declaration

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

### Returns

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

### Throws

CJMSException

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

## getJMSMessageID

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

### Declaration

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

### Returns

The message ID

### Throws

CJMSException

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

## getJMSPriority

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

### Declaration

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

### Returns

The default message priority

### Throws

CJMSEException

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

## getJMSRedelivered

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

### Declaration

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

### Returns

TRUE if this message is being redelivered

### Throws

CJMSEException

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

## getJMSReplyTo

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

### Declaration

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

## Returns

Destination to which to send a response to this message

## Throws

CJMSException

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

## getJMSTimestamp

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

## Declaration

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

## Returns

The timestamp of the message

## Throws

CJMSException

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

## getJMSType

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

## Declaration

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

## Returns

The message type

## Throws

CJMSException

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

## getLongProperty

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





















































































































































































































































































































































































































































