



**Fiorano**  
Enabling change at the speed of thought

[www.fiorano.com](http://www.fiorano.com)

# FioranoMQ® 9

## C RTL Guide

### AMERICA'S

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

### EMEA

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

### APAC

Fiorano Software Pte. Ltd.  
Level 42, Suntec Tower Three 8  
Temasek Boulevard 038988  
Singapore  
Tel: +65 68292234  
Fax: +65 68292235  
Email: [info\\_asiapac@fiorano.com](mailto:info_asiapac@fiorano.com)

**Fiorano**

Entire contents © Fiorano Software and Affiliates. All rights reserved. Reproduction of this document in any form without prior written permission is forbidden. The information contained herein has been obtained from sources believed to be reliable. Fiorano disclaims all warranties as to the accuracy, completeness or adequacy of such information. Fiorano shall have no liability for errors, omissions or inadequacies in the information contained herein or for interpretations thereof. The opinions expressed herein are subject to change without prior notice.

Copyright (c) 1999-2008, Fiorano Software Technologies Pvt. Ltd.,

Copyright (c) 2008-2009, Fiorano Software Pty. Ltd.

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.

# Content

---

<b>Chapter 1: Introduction.....</b>	<b>1</b>
Introduction to FioranoMQ CRTLS.....	1
Related Documentation .....	1
<b>Chapter 2: Data Types and Constants.....</b>	<b>2</b>
CRTL Data Types.....	2
CRTL Constants.....	3
Lookup Related Constants .....	3
Messaging Related Constants.....	4
<b>Chapter 3: Function Summary.....</b>	<b>6</b>
Lookup of Administered Objects.....	6
Creating InitialContext.....	6
HTTP support using the IntialContext.....	7
HTTPS Support using the InitialContext .....	8
SSL support using the InitialContext.....	9
Looking up Objects using InitialContext.....	10
Destroying the InitialContext .....	10
Point to Point Model.....	11
Using a QueueConnectionFactory .....	11
Creating a QueueConnection .....	11
Destroying a QueueConnectionFactory.....	11
Using a QueueConnection.....	12
Starting and Stopping a QueueConnection.....	12
ExceptionListener and ClientID of a QueueConnection.....	12
Creating a QueueSession.....	13
Closing and Destroying a QueueConnection .....	13
Using a QueueSession .....	14
Transactions on a QueueSession.....	14
Senders, Receivers and Browsers on a QueueSession .....	15
Creating Fiorano Messages.....	15
Creating and Destroying Queues.....	16
Asynchronous Listener on QueueSession.....	16
Closing and Destroying a QueueSession.....	16
Using a QueueSender .....	17
Message Params For A Sender.....	17
Sending a Message on a Queue .....	18

Closing and Destroying a QueueSender.....	19
Using a QueueReceiver .....	19
Synchronous Message Receive .....	19
Asynchronous Message Receive .....	20
Closing and Destroying a QueueReceiver .....	20
Request/Reply on Queues .....	21
Creating a Requestor .....	22
Sending a Request.....	22
Closing and Destroying a QueueRequestor .....	22
Browsing a Queue .....	22
Publish-Subscribe Model .....	23
Using a TopicConnectionFactory .....	23
Creating a TopicConnection.....	23
Destroying a TopicConnectionFactory .....	24
Using a TopicConnection .....	24
Starting and Stopping a TopicConnection .....	24
ExceptionListener on a TopicConnection.....	25
TopicSessions on a TopicConnection.....	25
Closing and Destroying a TopicConnection.....	26
Using a TopicSession.....	26
Transactions on a TopicSession .....	27
Publishers and Subscribers on a TopicSession .....	27
Creating Fiorano Messages.....	28
Creating and destroying Topics.....	28
Asynchronous Listener on TopicSession .....	29
Closing and Destroying a TopicSession .....	29
Using a TopicPublisher .....	29
Message Params for a Publisher.....	30
Publishing a message on a Topic.....	31
Destroying a TopicPublisher .....	31
Using a TopicSubscriber .....	31
Synchronous Message Receive .....	32
Asynchronous message receive .....	32
Destroying a TopicSubscriber .....	33
Request/Reply on Topics .....	33
Creating a TopicRequestor .....	34
Sending a request on a Topic .....	34
Closing and Destroying a TopicRequestor .....	35
Client-Side Logging.....	35
LH_setLoggerName .....	35
LH_getLogger.....	36
LH_setTraceLevel.....	36
LH_logData .....	36
CSPBrowser.....	36
CSPM_createCSPBrowser .....	37
CSPB_getAllConnections .....	37

CSPB_getTopicsForConnection .....	37
CSPB_getQueuesForConnection .....	37
CSPB_browseMessagesOnQueue .....	37
CSPB_browseMessagesOnQueue2 .....	37
CSPB_browseMessagesOnTopic .....	38
CSPB_browseMessagesOnTopic2 .....	38
CSPB_numberOfMessagesinCSPQueue .....	38
CSPB_numberOfMessagesinCSPTopic .....	38
CSPB_free .....	38
CSPE_hasMoreElements .....	38
CSPE_nextElement .....	39
CSPE_free .....	39
Fiorano Message .....	39
Message Bodies .....	40
Message Header .....	40
Message Properties .....	42
BytesMessage .....	43
MapMessage .....	45
StreamMessage .....	46
TextMessage .....	47
Acknowledging a message .....	47
Destroying a message .....	48
Large Message Support .....	48
LMSG_getMessageStatus .....	48
LMSG_setLMStatusListener .....	48
LMSG_getLMStatusListener .....	48
LMSG_saveTo .....	49
LMSG_resumeSaveTo .....	49
LMSG_resumeSend .....	49
LMSG_cancelAllTransfers .....	49
LMSG_cancelTransfer .....	49
LMSG_suspendAllTransfers .....	49
LMSG_suspendTransfer .....	50
LMSG_setFragmentSize .....	50
LMSG_getFragmentSize .....	50
LMSG_setWindowSize .....	50
LMSG_getWindowSize .....	50
LMSG_setRequestTimeoutInterval .....	50
LMSG_getRequestTimeoutInterval .....	50
LMSG_setResponseTimeoutInterval .....	51
LMSG_getResponseTimeoutInterval .....	51
LMTS_getBytesTransferred .....	51
LMTS_getBytesToTransfer .....	51
LMTS_getLastFragmentID .....	51
LMTS_getPercentageProgress .....	51
LMTS_getStatus .....	51

LM_TRANSFER_NOT_INIT or 1 .....	51
LM_TRANSFER_IN_PROGRESS or 2 .....	52
LM_TRANSFER_DONE or 3 .....	52
LM_TRANSFER_ERR or 4 .....	52
LM_ALL_TRANSFER_DONE or 5 .....	52
LMTS_isTransferComplete .....	52
LMTS_isTransferCompleteForAll .....	52
LMTS_getLargeMessage .....	52
LMTS_getConsumerID .....	52
LMC_getUnfinishedMessagesToSend .....	52
LMC_getUnfinishedMessagesToReceive.....	53
RME_hasMoreElements .....	53
RME_nextElement .....	53
Error Handling .....	53
Utility APIs .....	54
Key-value pairs in Hashtable .....	54

## Chapter 4: Function Reference ..... 55

getRuntimeVersion .....	55
DestroyHashtable .....	55
ES_free.....	55
FBMSG_clearBody .....	56
FBMSG_readBoolean.....	56
FBMSG_readByte .....	57
FBMSG_readBytes .....	57
FBMSG_readChar .....	58
FBMSG_readDouble .....	59
FBMSG_readFloat .....	59
FBMSG_readInt.....	60
FBMSG_readLong .....	60
FBMSG_readShort .....	61
FBMSG_readUnsignedByte.....	62
FBMSG_readUnsignedShort.....	62
FBMSG_readUTF .....	63
FBMSG_reset .....	63
FBMSG_writeBoolean .....	64
FBMSG_writeByte.....	64
FBMSG_writeBytes .....	65
FBMSG_writeBytesWithOffset .....	66
FBMSG_writeChar.....	66
FBMSG_writeDouble .....	67
FBMSG_writeFloat .....	67
FBMSG_writeInt .....	68
FBMSG_writeLong .....	69
FBMSG_writeShort .....	69

FBMSG_writeUTF .....	70
FMMSG_clearBody .....	70
FMMSG_getBoolean .....	71
FMMSG_getByte .....	72
FMMSG_getBytes .....	72
FMMSG_getChar .....	73
FMMSG_getDouble .....	73
FMMSG_getFloat .....	74
FMMSG_getInt .....	75
FMMSG_getLong .....	75
FMMSG_getMapNames .....	76
FMMSG_getShort .....	77
FMMSG_getString .....	77
FMMSG_itemExists .....	78
FMMSG_setBoolean .....	78
FMMSG_setByte .....	79
FMMSG_setBytes .....	80
FMMSG_setBytesWithOffset .....	80
FMMSG_setChar .....	81
FMMSG_setDouble .....	82
FMMSG_setFloat .....	82
FMMSG_setInt .....	83
FMMSG_setLong .....	84
FMMSG_setShort .....	84
FMMSG_setString .....	85
FSMSG_clearBody .....	86
FSMSG_readBoolean .....	86
FSMSG_readByte .....	87
FSMSG_readBytes .....	87
FSMSG_readChar .....	88
FSMSG_readDouble .....	89
FSMSG_readFloat .....	89
FSMSG_readInt .....	90
FSMSG_readLong .....	90
FSMSG_readShort .....	91
FSMSG_readString .....	92
FSMSG_reset .....	92
FSMSG_writeBoolean .....	93
FSMSG_writeByte .....	93
FSMSG_writeBytes .....	94
FSMSG_writeBytesWithOffset .....	95
FSMSG_writeChar .....	95
FSMSG_writeDouble .....	96
FSMSG_writeFloat .....	97
FSMSG_writeInt .....	97
FSMSG_writeLong .....	98

FSMSG_writeShort .....	98
FSMSG_writeString.....	99
FTMSG_getText.....	99
FTMSG_setText.....	100
HT_Clear.....	101
HT_ContainsKey.....	101
HT_ContainsValue .....	102
HT_Get .....	102
HT_IsEmpty .....	103
HT_Put .....	103
HT_RemoveElement.....	104
HT_Size .....	105
IC_Free.....	105
IC_Lookup.....	105
IC_LookupOCF .....	106
IC_LookupTCF .....	107
MqExceptionClear .....	107
MqExceptionOccurred.....	108
MqGetLastErrCode.....	108
MqPrintException.....	108
MqPrintExceptionToBuffer .....	109
MSG_acknowledge.....	109
MSG_clearBody.....	110
MSG_clearProperties .....	110
MSG_free .....	111
MSG_getBooleanProperty .....	111
MSG_getByteProperty .....	112
MSG_getDoubleProperty.....	113
MSG_getFloatProperty.....	113
MSG_getIntProperty .....	114
MSG_getJMSCorrelationID .....	115
MSG_getJMSCorrelationIDAsBytes .....	115
MSG_getJMSDeliveryMode.....	116
MSG_getJMSDestination.....	116
MSG_getJMSExpiration.....	117
MSG_getJMSMessageID .....	117
MSG_getJMSPriority.....	118
MSG_getJMSRedelivered .....	118
MSG_getJMSReplyTo.....	119
MSG_getJMSTimestamp .....	119
MSG_getJMSType.....	120
MSG_getLongProperty.....	120
MSG_getObjectProperty .....	121
MSG_getPropertyNames.....	122
MSG_getShortProperty.....	122
MSG_getStringProperty.....	123



MSG_propertyExists.....	123
MSG_setBooleanProperty.....	124
MSG_setByteProperty.....	125
MSG_setDoubleProperty.....	125
MSG_setFloatProperty.....	126
MSG_setIntProperty.....	126
MSG_setJMSCorrelationID.....	127
MSG_setJMSCorrelationIDAsBytes.....	128
MSG_setJMSDeliveryMode.....	128
MSG_setJMSDestination.....	129
MSG_setJMSExpiration.....	129
MSG_setJMSMessageID.....	130
MSG_setJMSPriority.....	131
MSG_setJMSRedelivered.....	131
MSG_setJMSReplyTo.....	132
MSG_setJMSTimestamp.....	132
MSG_setJMSType.....	133
MSG_setLongProperty.....	133
MSG_setObjectProperty.....	134
MSG_setShortProperty.....	135
MSG_setStringProperty.....	135
newHashtable.....	136
newInitialContext_env.....	136
newInitialContext_HTTP.....	137
newInitialContext_SSL.....	138
newInitialContext1.....	139
newInitialContext.....	139
newInitialContextDefaultParams.....	140
newQueueRequestor.....	141
newTopicRequestor.....	141
Q_free.....	142
Q_getQueueName.....	142
QBE_free.....	143
QBE_hasMoreElements.....	143
QBE_nextElement.....	143
QBWSR_close.....	144
QBWSR_free.....	144
QBWSR_getEnumeration.....	145
QBWSR_getMessageSelector.....	145
QBWSR_getQueue.....	146
QC_close.....	146
QC_createQueueSession.....	147
QC_free.....	148
QC_getClientID.....	148
QC_getExceptionListener.....	149
QC_setClientID.....	149

QC_setExceptionListener .....	150
QC_start .....	151
QC_stop.....	151
QCF_createQueueConnection .....	152
QCF_createQueueConnectionDefParams .....	152
QCF_free.....	153
QRCVR_close .....	153
QRCVR_free .....	154
QRCVR_getMessageListener.....	154
QRCVR_getMessageSelector .....	155
QRCVR_getQueue.....	155
QRCVR_receive .....	156
QRCVR_receiveNoWait .....	156
QRCVR_receiveWithTimeout.....	157
QRCVR_setFioranoMessageListener .....	157
QRCVR_setMessageListener .....	158
QRQST_close .....	159
QRQST_free .....	159
QRQST_request .....	159
QRQST_requestWithTimeout .....	160
QS_close.....	161
QS_commit .....	161
QS_createBrowser.....	162
QS_createBrowserWithSelector .....	162
QS_createBytesMessage.....	163
QS_createMapMessage.....	163
QS_createQueue .....	164
QS_createReceiver .....	164
QS_createReceiverWithSelector.....	165
QS_createSender .....	166
QS_createStreamMessage .....	166
QS_createTemporaryQueue .....	167
QS_createTextMessage .....	167
QS_createTextMessageWithText .....	168
QS_free .....	168
QS_getMessageListener .....	169
QS_getTransacted .....	169
QS_recover .....	170
QS_rollback.....	170
QS_setFioranoMessageListener.....	171
QS_setMessageListener.....	171
QSNDR_close.....	172
QSNDR_free .....	172
QSNDR_getDeliveryMode.....	173
QSNDR_getDisableMessageID .....	173
QSNDR_getDisableMessageTimestamp .....	174

QSNDR_getPriority .....	174
QSNDR_getQueue .....	175
QSNDR_getTimeToLive.....	176
QSNDR_send .....	176
QSNDR_send1 .....	177
QSNDR_send2 .....	178
QSNDR_send3 .....	178
QSNDR_setDeliveryMode.....	179
QSNDR_setDisableMessageID .....	180
QSNDR_setDisableMessageTimestamp .....	180
QSNDR_setPriority.....	181
QSNDR_setTimeToLive.....	181
T_free .....	182
T_getTopicName .....	182
TC_close .....	183
TC_createTopicSession.....	183
TC_free.....	184
TC_getClientID .....	184
TC_getExceptionListener .....	185
TC_setClientID.....	185
TC_setExceptionListener.....	186
TC_start.....	187
TC_stop .....	187
TCF_createTopicConnection .....	188
TCF_createTopicConnectionDefParams.....	188
TCF_free .....	189
TMPO_delete .....	189
TMPO_free .....	190
TMPT_delete .....	190
TMPT_free.....	191
TP_close .....	191
TP_free .....	192
TP_getDeliveryMode .....	192
TP_getDisableMessageID.....	193
TP_getDisableMessageTimestamp.....	193
TP_getPriority .....	194
TP_getTimeToLive .....	194
TP_getTopic.....	195
TP_publish .....	195
TP_publish1.....	196
TP_publish2.....	197
TP_publish3.....	197
TP_setDeliveryMode.....	198
TP_setDisableMessageID .....	199
TP_setDisableMessageTimestamp .....	199
TP_setPriority .....	200

TP_setTimeToLive.....	201
TRQST_close .....	201
TRQST_free .....	202
TRQST_request .....	202
TRQST_requestWithTimeout .....	203
TS_close .....	203
TS_commit .....	204
TS_createBytesMessage .....	204
TS_createDurableSubscriber .....	205
TS_createDurableSubscriberWithSelector .....	206
TS_createMapMessage .....	207
TS_createPublisher .....	207
TS_createPublisherOnTempTopic .....	208
TS_createStreamMessage .....	209
TS_createSubOnTempTopicWS .....	209
TS_createSubscriber .....	210
TS_createSubscriberOnTempTopic .....	210
TS_createSubscriberWithSelector .....	211
TS_createTemporaryTopic .....	212
TS_createTextMessage .....	212
TS_createTextMessageWithText .....	213
TS_createTopic .....	213
TS_free .....	214
TS_getMessageListener .....	214
TS_getTransacted .....	215
TS_recover .....	215
TS_rollback .....	216
TS_setFioranoMessageListener .....	216
TS_setMessageListener .....	217
TS_unsubscribe .....	218
TSUB_close .....	218
TSUB_free .....	219
TSUB_getMessageListener .....	219
TSUB_getMessageSelector .....	220
TSUB_getNoLocal .....	220
TSUB_getTopic .....	221
TSUB_receive .....	221
TSUB_receiveNoWait .....	222
TSUB_receiveWithTimeout .....	222
TSUB_setFioranoMessageListener .....	223
TSUB_setMessageListener .....	223
UCF_createConnectionWithParams .....	224
UCF_createConnection .....	225
CF_free .....	225
FC_createSession .....	226
FC_getClientID .....	226

FC_getExceptionListener .....	227
FC_setClientID .....	227
FC_setExceptionListener .....	228
FC_start .....	229
FC_stop .....	229
FC_free .....	230
FC_setUnifiedConnectionID .....	230
FC_getUnifiedConnectionID .....	231
US_getMessageListener .....	231
US_getTransacted .....	232
US_get AcknowledgeMode .....	232
US_setMessageListener .....	233
US_setFioranoMessageListener .....	233
US_createBrowser .....	234
US_createBrowserWithSelector .....	235
US_createConsumer .....	235
US_createConsumerWithSelector .....	236
US_createProducer .....	237
US_createDurableSubscriber .....	237
US_createDurableSubscriberWithSelector .....	238
US_createBytesMessage .....	239
US_createMapMessage .....	239
US_createStreamMessage .....	240
US_createTextMessage .....	240
US_createTextMessageWithText .....	241
US_commit .....	241
US_recover .....	242
US_rollback .....	242
US_close .....	243
US_free .....	243
US_unsubscribe .....	244
US_createTopic .....	244
US_createTemporaryTopic .....	245
US_createQueue .....	245
US_createTemporaryQueue .....	246
US_createQueueSender .....	247
US_createPublisher .....	247
FMP_send .....	248
FMP_sendWithParams .....	248
FMP_sendWithDestination .....	249
FMP_sendWithDestinationParams .....	250
FMP_close .....	251
FMP_free .....	251
FMP_getDeliveryMode .....	252
FMP_getDisableMessageID .....	252
FMP_getDisableMessageTimestamp .....	253

FMP_getPriority .....	253
FMP_getTimeToLive .....	254
FMP_setDeliveryMode .....	254
FMP_setDisableMessageID .....	255
FMP_setDisableMessageTimestamp .....	255
FMP_setPriority .....	256
FMP_setTimeToLive .....	257
FMC_close .....	257
FMC_free .....	258
FMC_start .....	258
FMC_stop .....	259

## Chapter 5: Using the C Runtime Library..... 260

Organization of Samples Provided .....	267
ptp .....	267
pubsub .....	267
Compiling and Running the Samples .....	0
On Windows .....	0
Compiling the samples .....	0
Running the Samples .....	1
On Solaris .....	1
Compiling the samples .....	1
Running the Samples .....	2

## Chapter 6: Native C-Runtime Examples..... 3

PTP Samples .....	4
Admin .....	4
Browser .....	4
msgsel .....	5
reqrep .....	5
basic .....	5
timeout .....	6
basic .....	6
Transactions .....	6
Http .....	7
Https .....	7
SSL .....	8
PubSub Samples .....	9
Admin .....	9
dursub .....	9
msgsel .....	10
basic .....	10
reqrep .....	11

basic .....	11
timeout.....	11
Transaction .....	12
Http.....	13
Https .....	13
ssl .....	14

# Chapter 1: Introduction

---

This guide provides an introduction to FioranoMQ C RunTime library. It also provides a summary of the various functions based on their usage and a detailed description of the APIs included in this library. Additionally, the guide also details out the steps involved in compiling and running CRTL sample applications.

## Introduction to FioranoMQ CRTLs

FioranoMQ C runtime library (CRTL) allows C and Java programs to communicate seamlessly. This version of CRTL supports both secure and non-secure TCP and HTTP connections on Win32 and Solaris platforms for Point-to-Point and Publish/Subscribe communication models. FioranoMQ provides C Runtime API library compatible with Microsoft Visual C++ and GNU C Compilers. The static Runtime Library provided by FioranoMQ is named 'fmq-crtl.lib' and for GNU C Compiler it is named "libfmq-crtl.a".

## Related Documentation

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

**Table 1 Related Documentation**

Document Name	Description
Native C++ Runtime Library Guide	Contains a description of FioranoMQ native C++ Runtime APIs
Native C# Runtime Library Guide	Contains a description of FioranoMQ C# Runtime APIs
JNI based C++ Runtime Library Guide	Provides an introduction to FioranoMQ C++ Runtime APIs
JavaDocs	Contains "javadoc style" documentation on all public Fiorano classes



# Chapter 2: Data Types and Constants

This chapter contains an overview of CRTL specific data types and constants.

## CRTL Data Types

The CRTL specific data types are defined in the *mq\_datatypes.h* file. This is done in order to make the communication format of the C client compliant with the Java based FioranoMQ server.

The most common data types used in CRTL are `mqbyte`, `mqchar`, `mqshort`, `mqint`, `mqlong`, `mqfloat`, `mqdouble`, `mqstring`, and `mqbyteArray`.

It is necessary to use these data types when programming with CRTL because of the difference in sizes of the corresponding data types in Java. The sizes of these data types are specified in the following table:

Data Type	Size
<code>mqbyte</code>	8 bits (unsigned)
<code>mqchar</code>	8 bits
<code>mqshort</code>	16 bits
<code>mqint</code>	32 bits
<code>mqlong</code>	64 bits
<code>mqfloat</code>	32 bits
<code>mqdouble</code>	64 bits

**TABLE 2-1 Size of Data Types**

The `mqstring` and `mqbyteArray` are defined as pointers to `mqchar` and `mqbyte`, respectively.

Apart from these basic data types, there is another important data type defined in CRTL "mqobject". It is defined to be (void \*). Certain functions, which can return different types of objects, returns an `mqobject` and it is the responsibility of the user to use the object in a suitable manne

It is also recommended to type cast the `mqobject` into an appropriate type, before using it.

The `mqchar` is defined to occupy only 8 bits and not 16 bits (as in Java), so that it is convenient to manage it in the user code.

## CRTL Constants

All the important public macros that are used in the CRTL are defined in the following header files:

1. initial\_context.h
2. common\_def.h

It is recommended to use these constants instead of using the corresponding values to avoid any complexities.

## Lookup Related Constants

Following are the permissible variable names and values in the InitialContext environment. This environment is used to create a new InitialContext, which in turn is used for looking up administered object from the FioranoMQ server.

Permissible names for the required variables:

- Username for accessing the admin store  
`#define SECURITY_PRINCIPAL "SECURITY_PRINCIPAL"`
- Password for the above mentioned username  
`#define SECURITY_CREDENTIALS "SECURITY_CREDENTIALS"`
- URL or location of the admin store (FioranoMQ server in this case)  
`#define PROVIDER_URL "PROVIDER_URL"`
- Protocol over which the C clients would communicate with the FioranoMQ server  
`#define TRANSPORT_PROTOCOL "TRANSPORT_PROTOCOL"`

Following are the permissible values for Transport protocol:

- `#define TCP_PROTOCOL "TCP"`
- `#define HTTP_PROTOCOL "HTTP"`
- `#define SSL_PROTOCOL "SSL"`
- `#define HTTPS_PROTOCOL "HTTPS"`

Following are the names of different variables that are used for client-server communication over HTTP protocol:

- URL of the HTTP proxy server through which the HTTP connections have to be routed  
`#define HTTP_PROXY_URL "HTTP_PROXY_URL"`
- The type of HTTP Proxy server being used between FioranoMQ client and server  
`#define HTTP_PROXY_TYPE "HTTP_PROXY_TYPE"`
- The authentication realms (if any) being used on the HTTP proxy server  
`#define PROXY_AUTHENTICATION_REALM "PROXY_AUTHENTICATION_REALM"`

- Username to pass through the HTTP Proxy server  
`#define PROXY_PRINCIPAL "PROXY_PRINCIPAL"`
- Password for the above mentioned username  
`#define PROXY_CREDENTIALS "PROXY_CREDENTIALS"`
- Variable for forcing HTTP 1.0 connections  
`#define FORCE_HTTP_10 "FORCE_HTTP_10"`

Following are names of different variables that are required for FioranoMQ client-server communication over SSL protocol:

- Fully qualified path with name of the SSL certificate file  
`#define SSL_CERT_FILE "SSL_CERT_FILE"`
- Fully qualified path with name of the SSL private key file  
`#define SSL_PRIVATE_KEY_FILE "SSL_PRIVATE_KEY_FILE"`
- Password for the SSL private key  
`#define SSL_PRIVATE_KEY_PASSWORD "`  
`SSL_PRIVATE_KEY_PASSWORD"`

## Messaging Related Constants

Messaging related constants are required while creating sessions and sending messages to a queue or topic on the FioranoMQ server.

Following are the possible values of the acknowledgement mode that can be used while creating a TopicSession or a QueueSession:

- `#define AUTO_ACKNOWLEDGE 1`
- `#define CLIENT_ACKNOWLEDGE 2`
- `#define DUPS_OK_ACKNOWLEDGE 3`

Following are the permissible values for the message delivery mode that can be used in publish or send call. The persistence of a message is decided using these constants:

- `#define NON_PERSISTENT 1`
- `#define PERSISTENT 2`

Following are the string representations of the various message types that can be sent by a C client to the FioranoMQ server. A call of `Msg_getJMSType` returns one of these values:

- `#define JMSMESSAGE "Message"`
- `#define BYTESMESSAGE "BytesMessage"`
- `#define STREAMMESSAGE "StreamMessage"`
- `#define MAPMESSAGE "MapMessage"`
- `#define TEXTMESSAGE "TextMessage"`

Following are some of the permissible values for message priority that can be used in publish or send call:

- `#define MinPriority 0`
- `#define MaxPriority 9`
- `#define LowPriority 0`
- `#define NormPriority 4`
- `#define HighPriority 9`

## Chapter 3: Function Summary

This chapter lists all the functions by category. If you know what you want to do, but do not know which function to use, look it up in this chapter. If you know the name of a function and want a complete description of it, you can find it in the chapter “Function Reference”.

### Lookup of Administered Objects

FioranoMQ supports lookup of administered objects using the JNDI interface. In case of the CRTL this support has been provided using an InitialContext structure that works mostly on the lines of the InitialContext object of JNDI. Client applications can create an InitialContext and lookup different administered objects from the FioranoMQ server.

### Creating InitialContext

The InitialContext is used to search for administered objects from the FioranoMQ server. The lookup operation performed by the InitialContext in the CRTL is similar to how it is done in Java using JNDI (Java Naming and Directory Interface). The only difference is that there is no entity called InitialContextFactory in the CRTL.

CRTL contains several APIs for creating the InitialContext. These can be sub-divided into two major categories:

- APIs that have individual parameters as arguments
- APIs that take a Hashtable environment as argument

In this case the different parameters required for creating the InitialContext like username, password, providerURL, and transport protocol are passed as separate arguments to the API. The table below lists these APIs with a brief purpose of each.

API	Function
newHashtable	Constructs a new, empty hashtable with a default initial capacity (11) and load factor, which is 0.75.
newInitialContext_HTTP	Constructs the Initial Context for lookup of Admin Objects, using the HTTP protocol for communication.
newInitialContext_SSL	Constructs the Initial Context for lookup of Admin Objects using the SSL protocol. The various SSL parameters are passed as arguments to this API.
newInitialContext	Constructs the Initial Context for lookup of Admin Objects using the url, username and password that are passed as parameters
newInitialContext1	Constructs the Initial Context for lookup of Admin Objects using the serverName, port, username and password passed as arguments.

API	Function
newInitialContextDefaultParams	Creates the Initial Context for lookup of Admin Objects using the default values for address, port number, user-name and password.

**APIs that take a Hashtable environment as argument**

The InitialContext can be created by passing all required environment parameters and their values in a Hashtable structure. The permissible name and values of some of these parameters can be checked in the chapter on datatypes and constants. For more information on creating and populating the hashtable structure, refer to the UtilityAPIs section of this chapter.

API	Function
newInitialContext_env	Constructs a new, empty hashtable with a default initial capacity (11) and load factor, which is 0.75.

**HTTP support using the InitialContext**

The HTTP mode allows client applications to connect to the FioranoMQ server run-ning in HTTP mode, both directly and through proxy servers. The only change re-quired in the application for using HTTP is while creating the InitialContext object. The following functions can be used to connect to the server, using the HTTP pro-tocol:

```
InitialContext newInitialContext_HTTP(mqstring url, mqstring user-name, mqstring passwd);
```

The above function is provided specifically for using HTTP protocol. However it does not allow you to specify a proxy server.

```
InitialContext newInitialContext_env(struct _Hashtable* env);
```

The above function is a more generic function, which accepts various parameters in a hashtable. This function can be used for all the protocols such as TCP, SSL, HTTP, and HTTPS. The following variables can be specified in the hashtable to configure the transport layer for HTTP:

- SECURITY\_PRINCIPAL Username, which is used to perform lookup at the FioranoMQ server.
- SECURITY\_CREDENTIALS Password for the user.
- PROVIDER\_URL URL of the FioranoMQ server to connect to
- TRANSPORT\_PROTOCOL Transport protocol to be used (HTTP in this case).
- HTTP\_PROXY\_URL URL of the proxy server to be used. If not specified, the runtime sends the request directly to the FioranoMQ server.
- HTTP\_PROXY\_TYPE Type of proxy server.
- PROXY\_AUTHENTICATION\_REALM Not supported currently.
- PROXY\_PRINCIPAL Username to be used for connecting to the proxy server.

- `PROXY_CREDENTIALS` Password for the proxy user.
- `FORCE_HTTP_10` Certain proxy servers do not support HTTP/1.1 persistent connections. For such servers, the flag "FORCE\_HTTP\_10" has to be set to TRUE.

The following code snippet has been extracted from an application using HTTP protocol.

```
InitialContext ic = NULL;
Hashtable env = newHashtable ();
HT_Put (env, SECURITY_PRINCIPAL, "anonymous");
HT_Put (env, SECURITY_CREDENTIALS, "anonymous");
HT_Put (env, PROVIDER_URL, "http://localhost:1856/");
HT_Put (env, HTTP_PROXY_URL, "http://localhost:8080");
HT_Put (env, HTTP_PROXY_TYPE, "MS_ISA_PROXY");
HT_Put (env, PROXY_AUTHENTICATION_REALM, "");
HT_Put (env, PROXY_PRINCIPAL, "modena");
HT_Put (env, PROXY_CREDENTIALS, "modena");
HT_Put (env, FORCE_HTTP_10, "TRUE")
ic = newInitialContext_env (env);
```

### HTTPS Support using the InitialContext

The current version of CRTL allows applications to connect to the FioranoMQ server, running in HTTP in the Phaos SSL mode. The following function can be used for configuring InitialContext to use HTTPS protocol.

```
InitialContext newInitialContext_env(struct _Hashtable* env);
```

The parameters that can be used for configuring C-runtime in HTTPS mode are as follows:

- `SECURITY_PRINCIPAL` Username, which is used to perform lookup at the FioranoMQ server.
- `SECURITY_CREDENTIALS` Password for the user.
- `PROVIDER_URL` URL of the FioranoMQ server to connect to.
- `TRANSPORT_PROTOCOL` Transport protocol to be used (HTTPS in this-case).
- `HTTP_PROXY_URL` URL of the proxy server to be used. If not specified, the runtime sends the request directly to the FioranoMQ server.
- `HTTP_PROXY_TYPE` Type of proxy server.
- `PROXY_AUTHENTICATION_REALM` Not supported currently.
- `PROXY_PRINCIPAL` Username to be used for connecting to the proxy server.
- `PROXY_CREDENTIALS` Password for the proxy user.
- `FORCE_HTTP_10` Certain proxy servers do not support HTTP/1.1 persistent connections. For such servers, the flag "FORCE\_HTTP\_10" has to be set-TRUE.
- `SSL_CERT_FILE` Fully qualified path of Certificate file.
- `SSL_PRIVATE_KEY_FILE` Fully qualified path of the Private key file.
- `SSL_PRIVATE_KEY_PASSWORD` Password for private key, if it is pass-word-encrypted, else NULL.

The following code snippet has been extracted from an application, which uses HTTPS protocol.

```
InitialContext ic = NULL;
Hashtable env = newHashtable ();
HT_Put (env, SECURITY_PRINCIPAL, "anonymous");
HT_Put (env, SECURITY_CREDENTIALS, "anonymous");
HT_Put (env, PROVIDER_URL, "http://localhost:1856/");
HT_Put (env, TRANSPORT_PROTOCOL, HTTPS_PROTOCOL);
HT_Put (env, HTTP_PROXY_URL, "http://localhost:8080");
HT_Put (env, HTTP_PROXY_TYPE, "MS_ISA_PROXY");
HT_Put (env, PROXY_AUTHENTICATION_REALM, "");
HT_Put (env, PROXY_PRINCIPAL, "modena");
```

### SSL support using the InitialContext

The only change in the C Client application for communicating with the FioranoMQ server on SSL is the creation of the InitialContext that is used for looking up various administered objects.

For a C client application to connect to the FioranoMQ server running over Phaos SSL, extra environment parameters are required for creating the InitialContext object. These extra parameters are listed below.

- **CertFile:** This is the digital certificate file. This parameter has to be passed with the fully qualified path name of the file. The certificate file must be in PEM format and must be sorted starting with the certificate to the highest level (root CA).
- **PrivateKeyFile:** This is the private key file. This parameter has to be passed-with the fully qualified path name of the file. The file must be in the PEM format.
- **keyPassword:** keyPassword is required if the private key file is encrypted with a password, otherwise a NULL value must be passed. These parameters can be provided to the InitialContext object in one of two possible ways.

These parameters can be added to a Hashtable structure that is provided in the CRTL. For creating a hashtable the `fiorano_hashtable.h` file needs to be included in the application. The Hashtable can be created using the following API.

```
Hashtable env = newHashtable ();
```

The parameters mentioned above can be added to this Hashtable in the following-manner.

```
HT_Put (env, TRANSPORT_PROTOCOL, SSL_PROTOCOL);
HT_Put (env, SSL_CERT_FILE, "C:\\Program Files\\Fiorano\\Fiora-noMQ\\crtl\\bin\\dsa-
client-
cert.der");
HT_Put (env, SSL_PRIVATE_KEY_FILE, " C:\\Program Files\\Fio-
rano\\FioranoMQ\\
crtl\\bin\\enc-dsa-client-key.pem");
HT_PUT(env, SSL_PRIVATE_KEYPASSWORD, "passwd");
```

Now this Hashtable is used for creating the InitialContext as follows:

```
InitialContext newInitialContext_env(struct _Hashtable* env);
```



This InitialContext can now be used to in the same way as it is for Client applications connecting over TCP.

Another way of creating the InitialContext is by using separate APIs for the case of SSL in which the above mentioned parameters are passed as arguments.

```
InitialContext newInitialContext_SSL(mqstring url, mqstring user-
name, mqstring passwd,
mqstring certFile, mqstring privateKeyFile, mqstring keyPass);
InitialContext newInitialContext1_SSL(mqstring serverName, mqint
port, mqstring username, mqstring passwd, mqstring certFile,
mqstring privateKeyFile, mqstring keyPass);
```

## Looking up Objects using InitialContext

A single API is provided for looking up any administered object from the FioranoMQ server. The name of the administered object is provided as an argument to the API. The API returns an mqobject that has to be type-casted into the structure of the administered object that is being looked up. The objects that can be looked up using the InitialContext are Topic, TopicConnectionFactory, Queue, and QueueConnectionFactory.

API	Function
IC_Lookup	Lookup an administered object from FioranoMQ server with name of the object passed as argument.
IC_LookupTCF	Lookup an administered object from FioranoMQ server with name of the object passed as argument. This is used for server less mode. If server is present, it looks up as normal Lookup function
IC_LookupQCF	Lookup an Queue Connection Factory object from FioranoMQ server with name of the object passed as argument. This is used for server less mode. If server is present, it looks up as normal Lookup function.

## Destroying the InitialContext

The InitialContext created in CRTL should be destroyed before the user program exits to avoid any memory leaks, using the following function.

API	Function
IC_free	Frees up the resources allocated to the initial context structure.

Following code snippet shows the use of the InitialContext.

```
InitialContext ic = NULL;
```

```
Queue queue = NULL;
Topic topic = NULL;
ic = newInitialContext("http://mymachine:1856",
"ayrton", "senna");
//...
queue = (Queue)IC_lookup(ic, "primaryqueue");
topic = (Topic)IC_lookup(ic, "primarytopic");
//...
IC_free(ic);
//...
//Free other objects before exit
```

## Point to Point Model

### Using a QueueConnectionFactory

The QueueConnectionFactory is a factory object used to create QueueConnections with the FioranoMQ server. A ConnectionFactory object encapsulates a set of connection configuration parameters that has been defined by an administrator. This is an administered object and should be looked up from the FioranoMQ server using the InitialContext (as explained in the section Lookup Administered Objects of this chapter). The user cannot create QueueConnectionFactory object using the CRTL. This can be done using the FioranoMQ AdminTool or by using the admin APIs provided in the default Java RTL.

One of the predefined QueueConnectionFactory objects in the FioranoMQ server is the primaryQCF. This is used in all the examples explained in the document where a QCF is necessary.

### Creating a QueueConnection

QueueConnectionFactories are used to create QueueConnections with the FioranoMQ server. The QueueConnection is created with the server running on the ConnectURL specified in the QueueConnectionFactory and if the same is unavailable then the RTL tries to make a connection with a BackupURL, if any.

QueueConnections can be created using default user identity ("anony-mous", "anonymous" in case of FioranoMQ) or by specifying a username and password.

API	Function
QCF_createQueueConnectionDefParams	Creates a queue connection with default user identity
QCF_createQueueConnection	Creates a queue connection with specified user identity

### Destroying a QueueConnectionFactory

Before quitting, the QueueConnectionFactory object, created in the application, should be destroyed using the following function.

API	Function
QCF_free	Frees all the resources allocated for the Queue-ConnectionFactory

The following code sample looks up a QueueConnectionFactory from the FioranoMQ server and uses it to create a QueueConnection.

```
QueueConnectionFactory qcf = NULL;
//...
qcf = (QueueConnectionFactory)IC_Lookup(ic, "primaryQCF");
qc = QCF_createQueueConnection(qcf, "ayrton", "senna");
//...
QCF_free(qcf);
//...
//Free other objects before exit.
```

## Using a QueueConnection

The QueueConnection is an active connection to the FioranoMQ server. It is used to create one or more queue sessions for producing and consuming messages. It is created using the QueueConnectionFactory. For more information, read the Using a QueueConnectionFactory section of this chapter.

## Starting and Stopping a QueueConnection

The delivery of messages on a connection is controlled by the start and stop APIs. Starting a connection starts or resumes the delivery of messages to consumers on that connection. Likewise, stopping the connection temporarily stops the delivery of messages to all consumers on that connection.

The ExceptionListener functionality is enabled only if the connection is started. The following APIs are used to control the start and stop of QueueConnections.

API	Function
QC_start	Starts (or restarts) the delivery of incoming messages of a connection.
QC_stop	Temporarily stops the delivery of incoming messages of a connection.

## ExceptionListener and ClientID of a QueueConnection

ExceptionListeners are used to check for error conditions on a QueueConnection. Users can set and get ExceptionListeners on a QueueConnection.

The ExceptionListener is not a structure but a callback function in the CRTL that is defined as:

```
void (*ExceptionListenerPointer)(char *exception, void *pParam);
```

The client applications have to set the function pointer of the above callback function as the ExceptionListener. When an exception occurs on a connection the same callback function is invoked by the CRTL. As can be seen from the above definition, CRTL gives the exception trace and a parameter as arguments to the callback function. The parameter in this case is set by the client application to be used by the user in the callback function.

The ClientID uniquely identifies a QueueConnection on the FioranoMQ server. If a ClientID has not been set, the FioranoMQ server assigns a unique ClientID to each QueueConnection.

<b>API</b>	<b>Function</b>
QC_setExceptionListener	Sets an exception listener for this connection
QC_getExceptionListener	Returns the ExceptionListener structure for this Queue connection
QC_setClientID	Sets the client identifier for this connection
QC_getClientID	Gets the client identifier for this connection

### Creating a QueueSession

A QueueConnection can be used to create QueueSessions using the following API.

<b>API</b>	<b>Function</b>
QC_createQueueSession	Creates a QueueSession with the given mode

### Closing and Destroying a QueueConnection

The QueueConnection has to be closed and destroyed before quitting the application.

Closing a QueueConnection terminates all pending messages received for all consumers created on the connection's sessions. Closing a connection not only closes the communication channel with the server but does not free up the memory allocated to the object. So, it must be destroyed. The following APIs are used for closing and destroying the QueueConnection.

API	Function
QC_close	Closes the connection.
QC_free	Frees all the resources allocated for the QueueConnection

Here is the sample code of how to use a QueueConnection and destroy it.

```
//...
QueueConnection qc = NULL;
//...
qc = QCF_createQueueConnection(qcf, "ayrton", "senna");
//...
QC_setClientID("myConnection");
QC_start(qc);
//...
QC_setExceptionListener(qc,onException,param);
//..
QC_createSession(qc,FALSE,AUTO_ACKNOWLEDGE);
QC_stop(qc);//if needed
//...
QC_close(qc);
//...
QC_free(qc);
//...
```

## Using a QueueSession

A QueueSession object provides methods for creating QueueReceiver, Queue-Sender, QueueBrowser, Queue, and TemporaryQueue objects.

If there are messages that have been received but not acknowledged when a QueueSession terminates, these messages are retained and redelivered when a consumer next accesses the queue.

## Transactions on a QueueSession

For non-transacted sessions any messages sent using the sender created on the session are sent to the server immediately following the QSNDR\_send() call and an acknowledgement is sent for the message, immediately when it is received.

JMS specifies another type of session named transacted session. In transacted sessions, each transaction groups a set of produced messages and a set of consumed messages into an atomic unit of work. When a transaction is committed, its atomic unit of input is acknowledged and its associated atomic unit of output is sent. If a transaction rollback is done, its produced messages are destroyed and its consumed messages are automatically recovered. A transacted session can be created by setting the value of the second argument to TRUE in the QC\_createSession API explained in the section "Using a QueueConnection" of this chapter.

The following APIs of CRTL are used to implement transacted sessions in applications.

API	Function
QS_commit	Commits all messages done in this transaction and releases any locks held at the time of call of this API.
QS_rollback	Rollbacks any messages done in this transaction and releases any locks held at the time of call of this API.
QS_getTransacted	Indicates if the session is in transacted mode
QS_recover	Stops message delivery in this session, and restarts message delivery with the oldest unacknowledged message

### Senders, Receivers and Browsers on a QueueSession

A QueueSession can be used to create senders, receivers, (with and without message selectors) and browsers (with and without message selectors) on a Queue using the following APIs.

API	Function
QS_createSender	Creates a QueueSender to send messages to the specified queue.
QS_createReceiver	Creates a QueueReceiver to receive messages from the specified queue.
QS_createReceiverWithSelector	Creates a QueueReceiver to receive messages from the specified queue, using a message selector
QS_createBrowser	Creates a QueueBrowser to peek at the messages using a message selector on the specified queue
QS_createBrowserWithSelector	Creates a QueueBrowser to peek at the messages using a message selector on the specified queue.

### Creating Fiorano Messages

A QueueSession can be used to create different message types using the APIs listed in the table below. All messages are created with the default JMSHeader and an empty message body.

API	Function
QS_createBytesMessage	Creates a bytes message.
QS_createMapMessage	Creates a map message.
QS_createStreamMessage	Creates a StreamMessage.
QS_createTextMessage	Creates a text message.
QS_createTextMessageWithText	Creates an initialized text message.

## Creating and Destroying Queues

The Queue is a destination object for messages in JMS. This is an administered object that can be looked up from the FioranoMQ server using the InitialContext. A C client application can also create its own Queue object using the QueueSession.

A QueueSession can be used to create a TemporaryQueue as well. The CRTL provides APIs to delete and destroy temporary queues as shown in the table below.

API	Function
QS_createQueue	Creates a queue identity given a Queue name.
QS_createTemporaryQueue	Creates a temporary queue
TMPO_delete	Deletes this temporary queue
Q_free	Frees all the resources allocated on behalf of a queue structure.
TMPO_free	Frees all the resources allocated on behalf of a temporary queue

## Asynchronous Listener on QueueSession

A QueueSession can be used to register a MessageListener on a Queue for receiving messages asynchronously. The delivery of messages on this listener starts only when a receiver is created on the Queue using the same QueueSession. For more information, read the Using a QueueReceiver section in this chapter.

The following APIs control the use of the MessageListener on a QueueSession.

API	Function
QS_setMessageListener	Sets the distinguished MessageListener of the session
QS_setFioranoMessageListner	Sets the distinguished MessageListener of the session. This API can be used to set a FioranoMessageListener callback pointer that can be passed a parameter, which in turn can be used in the implementation of the callback function.
QS_getMessageListener	Returns the distinguished message listener of the session.

## Closing and Destroying a QueueSession

QueueSessions need to be closed and destroyed before quitting an application

Closing a QueueSession terminates all pending message received for all consumers created on the connection's sessions. Closing a connection not only closes the communication channel with the server but does not free up the memory allocated to the object. So, it must be destroyed. The following APIs are used for closing and destroying the QueueConnection.

API	Function
QS_close	Closes the queue session and resources allocated on behalf of the QueueSession.
QS_free	Frees all the resources allocated on behalf of the QueueSession

## Using a QueueSender

A client uses a QueueSender to send messages to a queue. Normally, the Queue is specified when a QueueSender is created. This Queue can also be specified as NULL while creating a QueueSender in which case it is termed as an unidentified QueueSender.

If the QueueSender is created with an unidentified or NULL Queue, an attempt to use the send methods, which assume that the Queue has been identified, results in an error.

During the execution of its send method, a message must not be changed by other threads within the client. If the message is modified, the result of the send is undefined.

After sending a message, a client may retain and modify it without affecting the message that has been sent. The same message may be sent multiple times.

The following message headers are set as part of sending a message:

- JMSDestination
- JMSDeliveryMode
- JMSExpiration
- JMSPriority
- JMSMessageID
- JMSTimeStamp.

When the message is sent, the values of these headers are ignored. After the completion of the send, the headers hold the values specified by the method sending the message. It is possible for the send method not to set the JMSMessageID and JMSTimeStamp if the setting of these headers is explicitly disabled by the QSNDR\_setDisableMessageID or QSNDR\_setDisableMessageTimestamp method.

## Message Params For A Sender

CRTL provides APIs using which certain parameters in the MessageHeader of messages to be sent can be controlled at the QueueSender level. Applications can set the priority, delivery mode, and the time to live in the sender. These values are in turn be set for all messages that are sent by the concerned sender.

Applications can also disable the generation of JMSMessageID and JMSMessageTimestamp for all messages sent by a QueueSender.



The APIs for the above-mentioned functions are listed in the following table.

API	Function
QSNDR_setDeliveryMode	Sets the delivery mode for the sender.
QSNDR_getDeliveryMode	Gets the default delivery mode of the sender.
QSNDR_setPriority	Sets the default priority of the sender.
QSNDR_getPriority	Gets the default priority of the sender.
QSNDR_setTimeToLive	Sets the default length of time in milliseconds from its dispatch time, for which a produced message should be retained by the message system.
QSNDR_getTimeToLive	Gets the default length of time in milliseconds from its dispatch time, for which a produced message should be retained by the message system
QSNDR_setDisableMessageID	Sets whether message IDs are disabled.
QSNDR_getDisableMessageID	Gets an indication of whether message IDs are disabled.
QSNDR_setDisableMessageID	Sets whether message timestamps are disabled.
QSNDR_getDisableMessageTimestamp	Gets an indication of whether message timestamps are disabled

### Sending a Message on a Queue

The FioranoMQ CRTL provides different APIs for sending a message on a Queue. These APIs can be used to send messages on the Queue on which the sender is created and also on any other specified Queue.

API	Function
QSNDR_send	Sends a message to the queue on which the sender was created. Uses the default delivery mode, timeToLive and priority of the QueueSender
QSNDR_send1	Sends a message specifying delivery mode, priority and time to live to the queue.
QSNDR_send2	Sends a message to the specified queue. Uses the default delivery mode, timeToLive and priority of the QueueSender
QSNDR_send3	Sends a message to the specified queue, also specifying the delivery mode, priority and time to live to the queue

## Closing and Destroying a QueueSender

An application should close and destroy the QueueSender before quitting. The close call does not free the memory occupied by the QueueSender. All senders created using a QueueSession are also closed in the close call of the QueueSession.

API	Function
QSNDR_close	Closes the queue sender and the resources allocated on behalf of the QueueSender.
QSNDR_free	Frees the resources allocated on behalf of the queue sender

## Using a QueueReceiver

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

Although it is possible to have multiple QueueReceivers for the same queue, the CRTL API does not define how messages are distributed between the QueueReceivers.

If a QueueReceiver specifies a message selector, the messages that are not selected remain on the queue. By definition, a message selector allows a QueueReceiver to skip messages. This means that when the skipped messages are eventually read, the total ordering of the reads does not retain the partial order defined by each message producer. Only QueueReceivers without a message selector reads messages in message producer order.

## Synchronous Message Receive

Messages can be received synchronously from a Queue using the QueueReceiver's receive calls. These receive calls are of three distinct types:

- A blocking call that blocks for an infinite time period and returns as soon as a message is available in the queue
- A blocking call that blocks for a specified timeout period and returns if a message is available or if the timeout period expires
- A non-blocking call that returns without waiting for a message to be available

The APIs can be used to perform the synchronous receive operations mentioned above.

API	Function
QRCVR_receive	Receives the next message produced for this queue receiver.
QRCVR_receiveWithTimeout	Receives the next message that arrives within the specified timeout
QRCVR_receiveNoWait	Receives the next message if one is immediately available

## Asynchronous Message Receive

The CRTL specifies a way to receive messages asynchronously using the `MessageListener`. Unlike the JMS API, `MessageListener` is a callback function and not a structure. This callback function is defined as:

```
void (*MessageListener)(Message msg);
```

The CRTL provides another version of this callback function. This version can be given a parameter as function argument. This can be useful in cases where the application wants to perform some operation in the callback function that requires some external structure. This callback is termed as `FioranoMessageListener` and is defined as:

```
void (*FioranoMessageListener)(struct _Message*, void* pParam);
```

The callback function and the parameter are passed as arguments to the `setFioranoMessageListener` API described in the following table.

The `QueueSession` can also be used to set a `MessageListener` as explained in the section “Using a `QueueSession`” of this chapter. In case an application sets a `MessageListener` on both the `QueueSession` and a `QueueReceiver` created from the same session then the `QueueSession`'s callback function is invoked.

API	Function
<code>QRCVR_setMessageListener</code>	Sets <code>MessageListener</code> of the message consumer.
<code>QRCVR_setFioranoMessageListener</code>	Sets <code>MessageListener</code> of the message consumer. This API can be used to set a <code>FioranoMessageListener</code> callback pointer that can be passed a parameter, which in turn can be used in the implementation of the callback function.
<code>QRCVR_getMessageListener</code>	Gets the <code>MessageListener</code> of the queue receiver.

## Closing and Destroying a QueueReceiver

An application should close and destroy the `QueueReceiver` before quitting the application. The close call does not free the memory occupied by the `QueueReceiver`. All receivers created using a `QueueSession` are also closed in the close call of the `QueueSession`.

API	Function
<code>QRCVR_close</code>	Closes the queue receiver and the resources allocated on behalf of the <code>QueueReceiver</code>
<code>QRCVR_free</code>	Frees the resources assigned to a queue receiver

## Request/Reply on Queues

The QueueRequestor object is provided by the CRTL to simplify the making service requests. It creates a TemporaryQueue for the responses and provides a request function that sends the request message and waits for its reply. This request can be sent with a specified timeout interval as well.

The QueueReplier is not a separate structure in the CRTL like the QueueRequestor. A reply to a request message can be sent using the following steps.

1. Create a QueueReceiver on the Queue on which the requestor is sending requests. This QueueReceiver may receive messages synchronously or asynchronously.
2. Create a QueueSender on any Queue or a NULL Queue.
3. Start the QueueConnection on which the receiver is created.
4. When the request is received by the above QueueReceiver, get the JMSReplyTo (which is a TemporaryQueue by default) and the JMSCorrelationID (only for the case of requestWithTimeout) from the request message. For more information, read the Fiorano Message section in this chapter.
5. Now create a reply message using the QueueSession of the receiver and set the JMSCorrelationID extracted from the request in the reply message. This step has to be performed only if requestWithTimeout is being used.
6. Send the reply on the JMSReplyTo destination extracted from the request message using the QueueSender created above.

The implementation of the steps, mentioned above, is shown in the following code snippet.

```
qr = QS_createReceiver(qss, queue);

qsn = QS_createSender(qss, queue);
..//
snmsg = QS_createTextMessage(qss);
..//
QRCVR_setMessageListener(qr, onMsgRecv);
QC_start(qc);
void onMsgRecv(Message msg)
{
    rpstr = (mqstring)malloc(100*(sizeof(char)));
    mqstring corrID = NULL;
    Queue tqueue = NULL;
    ..//
    corrID = MSG_getJMSCorrelationID(msg);
    MSG_setJMSCorrelationID(snmsg, corrID);
    FTMSG_setText(snmsg, rpstr);
    ..//
    tqueue = MSG_getJMSReplyTo(msg);
    QSNDR_send2(qsn, tqueue, snmsg);
    ..// free the replyTo destination and the message structures}
```

## Creating a Requestor

The QueueRequestor is created on a Queue using a QueueSession. The following API can be used for the same.

API	Function
newQueueRequestor	Creates a new QueueRequestor on the specified Queue.

## Sending a Request

The QueueRequestor is used to send requests on a Queue. These requests can be sent with an infinite timeout or for a specified timeout period. In case of request with a timeout, the requestor waits for a reply for the timeout period and return null if no reply is available within the timeout period.

API	Function
QRQST_request	Sends a request and waits for a reply.
QRQST_requestWithTimeout	Sends a request and waits for a reply within the specified timeout interval

## Closing and Destroying a QueueRequestor

A QueueRequestor must be closed and destroyed before quitting an application. The close API of the QueueRequestor does not free the memory allocated to the structure.

API	Function
QRQST_close	Closes the queue requestor and all the resources allocated on behalf of the QueueRequestor.
QRQST_free	Frees the resources allocated on behalf of the QueueRequestor

## Browsing a Queue

A QueueBrowser is used for browsing a Queue for messages. This can be created using a QueueSession as explained in the section "Using a QueueSession" of this chapter.

A QueueBrowserEnumeration structure is returned by the getEnumeration call of the QueueBrowser which in turn is used for browsing the messages.

The QueueBrowser and QueueBrowserEnumeration have to be closed and destroyed by the application.

The following APIs are used for browsing a Queue.

API	Function
QBWSR_getQueue	Get the queue on which this queue browser is created
QBWSR_getMessageSelector	Gets message selector expression of this queue browser
QBWSR_getEnumeration	Gets an enumeration for browsing the current queue messages in the same order as they would be received
QBWSR_free	Frees the resources allocated for the Queue-Browser.
QBWSR_close	Closes all the resources on behalf of a Queue-Browser.
QBE_nextElement	Gets the next message available with the queue browser enumeration
QBE_hasMoreElements	Checks if the queue browser enumeration contains more elements that is, messages. QBE_free
QBE_free	Frees all the resources allocated on behalf of QueueBrowserEnumeration

## Publish-Subscribe Model

### Using a TopicConnectionFactory

TopicConnectionFactory is a factory object used to create TopicConnections with the FioranoMQ server. A ConnectionFactory object encapsulates a set of connection configuration parameters that has been defined by an administrator. This is an administered object and should be looked up from the FioranoMQ server using the InitialContext (as explained in the section Lookup Administered Objects of this chapter). The user cannot create his/her own TopicConnectionFactory object using the CRTL. This can be done using the FioranoMQ AdminTool or by using the admin APIs provided in the default Java RTL.

One of the predefined TopicConnectionFactory objects in the FioranoMQ server is the primaryTCF. This is used in all the examples explained in the document where a TCF is necessary.

### Creating a TopicConnection

TopicConnectionFactories are used to create TopicConnections with the FioranoMQ server. The TopicConnection is created with the server running on the ConnectURL specified in the TopicConnectionFactory and if the same is unavailable then the RTL tries to make a connection with a BackupURL, if any.

TopicConnections can be created using default user identity (“anonymous”, “anon-ymous” in case of FioranoMQ) or by specifying a username and password.

API	Function
TCF_createTopicConnection DefParams	Creates a queue connection with default user identity
TCF_createTopicConnection	Creates a queue connection with specified user identity.

### Destroying a TopicConnectionFactory

The TopicConnectionFactory object created in the application should be destroyed before quitting, using the following function.

API	Function
TCF_free	Frees all the resources allocated for the TopicConnectionFactory

. Here is the sample code of how to look up a TopicConnectionFactory from the FioranoMQ server and use it to create a TopicConnection.

```
TopicConnectionFactory tcf = NULL;
//...
tcf = (TopicConnectionFactory)IC_Lookup(ic, "primaryTCF");
tc = QCF_createTopicConnection(tcf, "ayrton", "senna");
//...
TCF_free(tcf);
//...
//Free other objects before exit.
```

### Using a TopicConnection

The TopicConnection is an active connection to the FioranoMQ server. It is used to create one or more topic sessions for producing and consuming messages. It is created using the TopicConnectionFactory as mentioned in the “Using a TopicConnection-Factory” section of this chapter.

### Starting and Stopping a TopicConnection

The delivery of messages on a connection is controlled by the start and stop APIs. Starting a connection starts or resumes the delivery of messages to consumers on that connection and stopping the connection temporarily stops the delivery of messages to all consumers on that connection.

The ExceptionListener functionality also gets enabled only if the connection is started.

Following APIs are used to control the start and stop of TopicConnections.

API	Function
TC_start	Starts (or restarts) delivery of incoming messages of a connection
TC_stop	Temporarily stops delivery of incoming messages of a connection.

### ExceptionListener on a TopicConnection

ExceptionListeners are used to check for error conditions on a TopicConnection. Users can set and get ExceptionListeners on a TopicConnection.

The ExceptionListener is not a structure but a callback function in the CRTL that is defined as

```
void (*ExceptionListenerPointer)(char *exception, void *pParam);
```

The client applications have to set the function pointer for such a callback function as the ExceptionListener and whenever some exception occurs on a connection that callback function is invoked by the CRTL. As can be seen from the above definition the CRTL gives the exception trace and a parameter as arguments to the callback function. The parameter in this case is set by the client application to be used by the user in the callback function.

The ClientID uniquely identifies a TopicConnection on the FioranoMQ server. If no ClientID is set the FioranoMQ server assigns a unique ClientID to each TopicConnection.

API	Function
TC_setExceptionListener	Sets an exception listener for this connection
TC_getExceptionListener	Gets the ExceptionListener object for this connection
TC_setClientID	Sets the client identifier for this connection
TC_getClientID	Gets the client identifier for this connection.

### TopicSessions on a TopicConnection

A TopicConnection can be used to create TopicSessions using the following API.

API	Function
TC_createTopicSession	Creates a TopicSession with the given mode.



## Closing and Destroying a TopicConnection

The TopicConnection has to be closed and destroyed before quitting from the user application.

Closing a TopicConnection terminates all pending message receives on the connection's session's consumers. Closing a connection only closes the communication channel with the server but doesn't free up the memory allocated to the object. So, it must be destroyed. The following APIs are used for closing and destroying the TopicConnection.

API	Function
TC_close	Terminates the connection with Fiorano JMS server and closes the resources allocated on behalf of the TopicConnection
TC_free	Frees all the resources allocated for the TopicConnection

Here is the sample code of how to use a TopicConnection and destroy it.

```
//...
TopicConnection qc = NULL;
//...
tc = TCF_createTopicConnection(tcf, "ayrton", "senna");
//...
TC_setClientID("myConnection");
TC_start(tc);
//...
TC_setExceptionListener(qc,onException,param);
//..
TC_createSession(tc,FALSE,AUTO_ACKNOWLEDGE);
TC_stop(tc);//if needed
//...
TC_close(tc);
//...
TC_free(tc);
//...
```

## Using a TopicSession

A TopicSession provides methods for creating TopicPublishers, TopicSubscribers, and TemporaryTopics. It also provides the unsubscribe method for deleting its client's durable subscriptions. If there are messages that have been received but not acknowledged when a TopicSession terminates, a durable TopicSubscriber must retain and redeliver them; a nondurable subscriber need not do so.

## Transactions on a TopicSession

For non-transacted sessions any messages sent using the publisher created on the session are sent to the server immediately following the `TP_publish()` call and an acknowledgement is sent for the message, immediately when it is received by a subscriber.

JMS specifies another type of session that is, transacted session, in which each transaction groups a set of produced messages and a set of consumed messages into an atomic unit of work. When a transaction is committed, its atomic unit of input is acknowledged and its associated atomic unit of output is sent. If a transaction rollback is done, its produced messages are destroyed and its consumed messages are automatically recovered. A transacted session can be created by setting the value of the second argument to `TRUE` in the `TC_createTopicSession` API explained in the section “Using a TopicConnection” of this chapter.

The following APIs of CRTL are used to implement transacted sessions in applications.

API	Function
TS_commit	Commits all messages done in this transaction and releases any locks held at the time of call of this API.
TS_rollback	Rollbacks any messages done in this transaction and releases any locks held at the time of call of this API.
TS_getTransacted	Gets an indication whether the session is transacted.
TS_recover	Stops message delivery in this session, and restarts sending messages with the oldest unacknowledged message.

## Publishers and Subscribers on a TopicSession

A TopicSession can be used to create publishers and subscribers (with and without message selectors) on a Topic using the following APIs.

Both durable and non-durable subscribers can be created using a TopicSession.

API	Function
TS_createPublisher	Creates a Publisher for the specified topic.
TS_createSubscriber	Creates a non-durable Subscriber to the specified topic.
TS_createSubscriber OnTempTopic	Creates a non-durable Subscriber to the specified temporary topic
TS_createSubscriber WithSelector	Creates a non-durable Subscriber to the specified topic using the specified message selector
TS_createDurableSub scriber	Creates a durable Subscriber to the specified topic.

API	Function
TS_createDurableSubscriberWithSelector	Creates a Durable Subscriber to the given Topic with given message selector and local message delivery options

### Creating Fiorano Messages

A TopicSession can be used to create different message types using APIs listed in the table below. All messages are created with the default JMSHeader and an empty message body.

API	Function
TS_createBytesMessage	Creates a bytes message
TS_createMapMessage	Creates a map message
TS_createStreamMessage	Creates a StreamMessage
TS_createTextMessage	Creates a text message
TS_createTextMessageWithText	Creates an initialized text message

### Creating and destroying Topics

The Topic is a destination object for messages in the publish/subscribe model of JMS. This is an administered object that can be looked up from the FioranoMQ server using the InitialContext. A C client application can also create its own Topic object using the TopicSession.

Before quitting a C client application both Topics and Temporary topics must be freed if created or looked up.

API	Function
TS_createTopic	Creates a topic identity given a topic name
TS_createTemporaryTopic	Creates a temporary topic
TMPT_delete	Deletes this temporary topic
T_free	Frees the memory allocated to the topic structure
TMPT_free	Frees all the resources allocated on behalf of the temporary topic

## Asynchronous Listener on TopicSession

A TopicSession can be used to register a MessageListener on a Topic for receiving messages asynchronously. The delivery of messages on this listener starts only when a subscriber is created on the Topic using the same TopicSession. For more information on the MessageListener refer to the section “Using a TopicSubscriber” in this chapter.

The following APIs control the use of the MessageListener on a TopicSession.

API	Function
TS_setMessageListener	Sets the distinguished message listener of the session.
TS_setFioranoMessageListener	Sets the distinguished MessageListener of the session. This API can be used to set a FioranoMessageListener callback pointer that can be passed a parameter, which in turn can be used in the implementation of the callback function.
TS_getMessageListener	Returns the distinguished message listener of the session.

## Closing and Destroying a TopicSession

TopicSessions need to be closed and destroyed before quitting an application that is using them.

Closing a TopicSession terminates all pending message receives on the session’s consumers. Closing a session doesn’t free up the memory allocated to the object,

API	Function
TS_close	Closes the topic session and resources allocated on behalf of the TopicSession
TS_free	Frees all the resources allocated on behalf of the TopicSession

## Using a TopicPublisher

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

Normally, the Topic is specified when a TopicPublisher is created. This Topic can also be specified as NULL while creating a TopicPublisher in which case it is termed as an unidentified TopicPublisher.

If the TopicPublisher is created with an unidentified or NULL Topic, an attempt to use the send methods that assume that the Topic has been identified results in an error.

During the execution of its publish method, a message must not be changed by other threads within the client. If the message is modified, the result of the publish is undefined.

After publishing a message, a client may retain and modify it without affecting the message that has been published. The same message object may be published multiple times.

The following message headers are set as part of publishing a message: JMSDestination, JMSDeliveryMode, JMSExpiration, JMSPriority, JMSMessageID and JMSTimeStamp. When the message is published, the values of these headers are ignored. After completion of the publish, the headers hold the values specified by the method publishing the message. It is possible for the publish method not to set JMSMessageID and JMSTimeStamp if the setting of these headers is explicitly disabled by the TP\_setDisableMessageID or TP\_setDisableMessageTimestamp method.

### Message Params for a Publisher

The CRTL provides APIs using which certain parameters in the MessageHeader of messages to be published can be controlled at the TopicPublisher level. Applications can set the priority, delivery mode and the time to live in the publisher. These values are in turn be set for all messages that are published by the concerned publisher.

Applications can also disable the generation of JMSMessageID and JMSMessageTimestamp for all messages published by a TopicPublisher.

API	Function
TP_setDeliveryMode	Sets the default delivery mode of the producer
TP_getDeliveryMode	Gets the default delivery mode of the producer
TP_setPriority	Sets the default priority of the producer
TP_getPriority	Gets the default priority of the producer
TP_setTimeToLive	Sets the default length of time in milliseconds from its dispatch time, for which a produced message should be retained by the message system.
TP_getTimeToLive	Gets the default length of time in milliseconds, from its dispatch time, for which a produced message should be retained by the message system
TP_setDisableMessageID	Sets whether message IDs are disabled for this Topic Publisher
TP_getDisableMessageID	Gets an indication of whether message IDs are disabled.
TP_setDisableMessageTimestamp	Sets whether message timestamps are disabled
TP_getDisableMessageTimestamp	Gets an indication of whether message timestamps are disabled

## Publishing a message on a Topic

The FioranoMQ CRTL provides different APIs for publishing a message on a Topic. These APIs can be used to publish messages on the Topic on which the publisher is created and also on any other specified Topic.

API	Function
TP_publish	Publishes a message to the topic on which the publisher is created. Uses the default delivery mode, timeToLive and priority of the topic
TP_publish1	Publishes a message to the topic on which the publisher is created, specifying delivery mode, priority and time to live to the topic.
TP_publish2	Publishes a message to a topic for an unidentified message producer. Uses the default delivery mode, timeToLive and priority of the topic.
TP_publish3	Publishes a message to a topic for an unidentified message producer, specifying delivery mode, priority and time to live.

## Destroying a TopicPublisher

An application should close and destroy the TopicPublisher before quitting the application. The close call does not free the memory occupied by the TopicPublisher. All senders created using a TopicSession are also closed in the close call of the TopicSession.

API	Function
TP_close	Closes the TopicPublisher and the resources allocated on behalf of the TopicPublisher.
TP_free	Frees the resources allocated on behalf of the topic publisher

## Using a TopicSubscriber

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

A TopicSession allows the creation of multiple TopicSubscriber objects per topic. It delivers each message for a topic to each subscriber eligible to receive it. Each copy of the message is treated as a completely separate message. Work done on one copy has no effect on the others; acknowledging one does not acknowledge the others; one message may be delivered immediately, while another waits for its subscriber to process messages ahead of it.

Regular TopicSubscriber objects are not durable. They receive only messages that are published while they are active.

Messages filtered out by a subscriber's message selector would never be delivered to the subscriber. From the subscriber's perspective, they do not exist.

In some cases, a connection may both publish and subscribe to a topic. The subscriber NoLocal attribute allows a subscriber to inhibit the delivery of messages published by its own connection.

If a client needs to receive all the messages published on a topic, including the ones published while the subscriber is inactive, it uses a durable TopicSubscriber. The FioranoMQ server retains a record of this durable subscription and insures that all messages from the topic's publishers are retained until they are acknowledged by this durable subscriber or they have expired.

Sessions with durable subscribers must always provide the same client identifier. In addition, each client must specify a name that uniquely identifies (within client identifier) each durable subscription it creates. Only one session at a time can have a TopicSubscriber for a particular durable subscription.

A client can change an existing durable subscription by creating a durable Topic-Subscriber with the same name and a new topic and/or message selector. Changing a durable subscription is equivalent to unsubscribing (deleting) the old one and creating a new one.

The unsubscribe method is used to delete a durable subscription. The unsubscribe method can be used at the Session or TopicSession level. This method deletes the state being maintained on behalf of the subscriber by its provider.

### Synchronous Message Receive

Messages can be received synchronously from a Topic using the receive call of the TopicSubscriber. This receive call is a blocking call that blocks for an infinite time period, for a specified timeout period or does not block at all. The call returns as soon as a message is available on the concerned Topic. The following APIs can be used to perform the synchronous receive operations.

API	Function
TSUB_receive	Receives the next message produced for this topic subscriber
TSUB_receiveWithTime	Receives the next message that arrives within the out specified timeout interval.
TSUB_receiveNoWait	Receives the next message if one is immediately available

### Asynchronous message receive

The CRTL specifies a way to receive messages asynchronously using the MessageListener. Unlike the JMS API the MessageListener in the CRTL is a callback function and not a structure. This callback function is defined as

```
void (*MessageListener)(Message msg);
```

This callback is invoked by the CRTL when a message arrives from the FioranoMQ server. A client application has to set the function pointer of the callback function it declares in the setMessageListener call.

The CRTL provides another version of this callback function that can be given a parameter as function argument. This can be useful in cases where the application wants to perform some operation in the callback function that requires some external structure. This callback is termed as FioranoMessageListener and is defined as

```
void (*FioranoMessageListener)(struct _Message*, void* pParam);
```

The callback function and the parameter are passed as arguments to the setFioranoMessageListener API described in the table below.

A TopicSession can also be used to set a MessageListener as explained in the section “Using a TopicSession” of this chapter. In case an application sets a Message.

API	Function
TSUB_setMessageListner	Sets the MessageListener of the message consumer
TSUB_setFioranoMessageListener	Sets MessageListener of the message consumer. This API can be used to set a FioranoMessageListener callback pointer that can be passed a parameter, which in turn can be used in the implementation of the callback function.
TSUB_getMessageListner	Gets the MessageListener of this message consumer

### Destroying a TopicSubscriber

An application should close and destroy the TopicSubscriber before quitting the application. The close call does not free the memory occupied by the TopicSubscriber. All subscribers created using a TopicSession are also closed in the close call of the TopicSession.

API	Function
TSUB_close	Closes the TopicSubscriber and the resources allocated on behalf of TopicSubscriber.
TSUB_free	Frees the resources allocated on behalf of Topic-Subscriber

### Request/Reply on Topics

The TopicRequestor object is provided by the CRTL to simplify making service requests. It creates a TemporaryTopic for the responses and provides a request function that sends the request message and waits for its reply. This request can be sent with a specified timeout interval as well.

The TopicReplier is not a separate structure in the CRTL like the TopicRequestor. A reply to a request message on a Topic can be sent using the following steps.

1. Create a TopicSubscriber on the Topic on which the requestor is sending requests. This TopicSubscriber may receive messages synchronously or asynchronously.
2. Create a TopicPublisher on any Topic or a NULL Topic.
3. Start the TopicConnection on which the subscriber is created.
4. When the request is received by the above TopicSubscriber get the JMSReplyTo (which is a TemporaryTopic) and the JMSCorrelationID (only for the case of requestWithTimeout) from the request message. For information on APIs to get the mentioned parameters refer to the section “FioranoMessage” in this chapter.



5. Now create a reply message using the TopicSession of the subscriber and set the JMSCorrelationID extracted from the request in the reply message. This step has to be done only if requestWithTimeout is being used.
6. Send the reply on the JMSReplyTo destination extracted from the request message using the TopicPublisher created above.

The following code snippet depicts the above mentioned steps.

```

..// steps for creation of TopicSession and lookup of Topic
tsub = TS_createSubscriber(ts, topic);
tpub = TS_createPublisher(ts, topic);
..//
snmsg = TS_createTextMessage(ts);
..//
TSUB_setMessageListener(tsub, onMsgRecv);
TC_start(tc);
void onMsgRecv(Message msg)
{
rpstr = (mqstring)malloc(100*(sizeof(char)));
mqstring corrID = NULL;
Topic tTopic = NULL;
..//
corrID = MSG_getJMSCorrelationID(msg);
MSG_setJMSCorrelationID(snmsg, corrID);
FTMSG_setText(snmsg, rpstr);
..//
tTopic = MSG_getJMSReplyTo(msg);
TPUB_publish2(ts, tTopic, snmsg);
..// free the replyTo destination and the message structures

```

### Creating a TopicRequestor

The TopicRequestor is created on a Topic using a TopicSession. The following API can be used for the same.

API	Function
newTopicRequestor	Creates a TopicRequestor

### Sending a request on a Topic

The TopicRequestor is used to send requests on a Topic. These requests can be sent with an infinite timeout or for a specified timeout period. In case of request with

API	Function
TRQST_request	Sends a request and waits for a reply
TRQST_requestWithTimeout	Sends a request and waits for a reply within the specified timeout interval

## Closing and Destroying a TopicRequestor

A TopicRequestor must be closed and destroyed before quitting an application. The close API of the TopicRequestor does not free the memory allocated to the structure.

API	Function
TROST_close	Closes the topic requestor and all the resources allocated on behalf of TopicRequestor.
TROST_free	Frees the resources allocated on behalf of TopicRequestor

## Client-Side Logging

In order to enable logging in CRTL, the following system environmental variables have to be set.

Variable	Value
ENABLE_CLIENT_LOGGER	TRUE
TRACE_LEVEL	ERROR, INFO, DEBUG
ERROR	logs only when error has occurred.
INFO	logs general information & errors
DEBUG	logs general information, errors and debug statements

In INFO level information about successful look up or creation of objects is logged. Logging in publish and receive calls are moved to DEBUG level due to the file size constraint when messages are published continuously.

- In DEBUG level, addresses of any memory that is being freed are logged.
- If DEBUG level is set, both INFO and ERROR statements will be logged.
- Similarly, if INFO level is set ERROR statements will also be logged.

Logging is done internally in CRTL. The log file will be created in the directory where the application exists. The following functions are available for the user.

### LH\_setLoggerName

**Declaration:** mqboolean LH\_setLoggerName(mqstring name);

sets file name[in which data is logged] to name. The default file name is "cclient".

For example --

```
LH_setLoggerName("Publisher");
```

'log' is appended to the file name.

**Note:** The maximum file size for the log file is set to 1 MB. As a result new log file will be created when the file size exceeds 1 MB. New files will be created with numbering as name\_0.log, name\_1.log, name\_2.log and so on.

## LH\_getLogger

**Declaration:** LogHandler \*LH\_getLogger();

It returns the reference to the logHandler, using which data can be logged from application.

The structure used for logger is LogHandler.

## LH\_setTraceLevel

**Declaration:** mqboolean LH\_setTraceLevel(LogHandler logger, mqstring level);

The trace level can also be set using this function. The legal values for the "level" are:

1. "ERROR"
2. "INFO"
3. "DEBUG"

For example, trace level can be set or changed to "INFO" by calling, LH\_setTraceLevel(logger, "INFO");

## LH\_logData

**Declaration:** mqboolean LH\_logData(LogHandler logger, mqstring info, ...);

This function logs the given data to the log file. It is variable argument function. The usage of this function is similar to "printf" function.

For example, LH\_logData(logger, "Logging from file - %s, func -%s", "FileName", "funcName");

In log file it is printed as follows:

```
Fri Dec 19 14:41:25 2008 :APPL: Logging from file - FileName, func -funcName
```

## CSPBrowser

CSP Browser is used to browse the messages stored in Client-Side Persistent Cache. While browsing messages are read from CSP and are not deleted. So, next time when the Server is re-connected, messages are sent to the Server.

The structures used for CSP Browsing are CSPBrowser and CSPEnumeration.

The following functions are used for CSP Browsing:

### CSPM\_createCSPBrowser

**Declaration:** CSPBrowser CSPM\_createCSPBrowser(mqstring dbPath);

This function returns the CSPBrowser Object for CSP cache located in the dbPath.

### CSPB\_getAllConnections

**Declaration:** ListEnumerator CSPB\_getAllConnections(CSPBrowser browser);

This function returns the enumeration of all the Connection IDs whose messages are stored in CSP

### CSPB\_getTopicsForConnection

**Declaration:** ListEnumerator CSPB\_getTopicsForConnection(CSPBrowser browser, mqstring connID);

This function returns the Enumeration of all the topic names for the specified connection with clientID as connID.

### CSPB\_getQueuesForConnection

**Declaration:** ListEnumerator CSPB\_getQueuesForConnection(CSPBrowser browser, mqstring connID);

This function returns the Enumeration of all the queue names for the specified connection with clientID as connID.

### CSPB\_browseMessagesOnQueue

**Declaration:** CSPEnumeration CSPB\_browseMessagesOnQueue(CSPBrowser browser, mqstring queueName, mqboolean checkTransacted);

This function returns the enumeration of messages stored in the queue specified by queueName. It searches for messages in queue in all the existing connections. If **checkTransacted** is TRUE, transacted messages are also browsed.

### CSPB\_browseMessagesOnQueue2

**Declaration:** CSPEnumeration CSPB\_browseMessagesOnQueue2(CSPBrowser browser, mqstring connectionID, mqstring queueName, mqboolean checkTransacted);

This function returns the enumeration of messages stored in the queue specified by queueName. It searches for messages in queue for the connection specified by connectionID. If **checkTransacted** is TRUE, transacted messages are also browsed.

## CSPB\_browseMessagesOnTopic

**Declaration:** CSPEnumeration CSPB\_browseMessagesOnTopic(CSPBrowser browser, mqstring topicName, mqboolean checkTransacted);

This function returns the enumeration of messages stored in the topic specified by topicName. It searches for messages in topic in all the existing connections. If **checkTransacted** is TRUE, transacted messages are also browsed.

## CSPB\_browseMessagesOnTopic2

**Declaration:** CSPEnumeration CSPB\_browseMessagesOnTopic2(CSPBrowser browser, mqstring connectionID, mqstring topicName, mqboolean checkTransacted);

This function returns the enumeration of messages stored in the topic specified by topicName. It searches for messages in topic for the connection specified by connectionID. If **checkTransacted** is TRUE, transacted messages are also browsed.

## CSPB\_numberOfMessagesinCSPQueue

**Declaration:** mqlong CSPB\_numberOfMessagesinCSPQueue(CSPBrowser browser, mqstring connID, mqstring queueName, mqboolean checkTransacted);

This function returns the number of messages stored in the queue specified by queueName for a given connection with clientID connID.

If **checkTransacted** is TRUE, transacted messages are also counted.

## CSPB\_numberOfMessagesinCSPTopic

**Declaration:** mqlong CSPB\_numberOfMessagesinCSPTopic(CSPBrowser browser, mqstring connID, mqstring topicName, mqboolean checkTransacted);

This function returns the number of messages stored in the topic specified by topicName for a given connection with clientID connID.

If **checkTransacted** is TRUE, transacted messages are also counted.

## CSPB\_free

**Declaration:** void CSPB\_free(CSPBrowser browser);

This function frees the CSP Browser Structure.

## CSPE\_hasMoreElements

**Declaration:** mqboolean CSPE\_hasMoreElements(struct \_CSPEnumeration\* cspEnum);

This function checks whether more elements are available in the enumeration.

## CSPE\_nextElement

**Declaration:** mqobject CSPE\_nextElement(struct \_CSPEnumeration\* cspEnum);

This function returns the next element of this enumeration if this enumeration object has at least one more element to provide.

## CSPE\_free

**Declaration:** void CSPE\_free(struct \_CSPEnumeration\* cspEnum);

This function frees the CSP Enumeration structure cspEnum.

## Fiorano Message

Message is the most widely used entity in JMS. The different types of messages defined by JMS are:

- TextMessage
- StreamMessage
- BytesMessage
- MapMessage
- ObjectMessage

The CRTL provides the functionality related to TextMessage, StreamMessage, BytesMessage, and MapMessage. The ObjectMessage functionality is not provided.

Apart from these four types, a generic message type "Message" is also defined. In the CRTL, all the four of them are just synonyms for the same type, "Message".

The type name Message is used in the JMS functions, which can operate on any type of message. The specific type names are used in the functions, which operate on that particular type of message.

The methods defined in the base class Message and its subclasses in JMS are defined in CRTL with meaningful prefixes. These prefixes are summarized in the following table.

<b>Message</b>	<b>Prefix</b>
Message	MSG
TextMessage	FTMSG
BytesMessage	FBMSG
StreamMessage	FSMSG
MapMessage	FMMSG

Even though these functions expect specific types of messages as arguments, the compiler won't detect any errors if different types of messages are passed. This is because all messages are just type defined to the same type. Therefore, it is the responsibility of the developer, to send proper types of messages as arguments, failing which an error is set at the runtime.

JMS messages are composed of the following parts:

- **Header** All messages support the same set of header fields. Header fields contain values used by both clients and providers to identify and route messages.
- **Properties** Each message contains a built-in facility for supporting application-defined property values. Properties provide an efficient mechanism for supporting application-defined message filtering.
- **Body** The JMS API defines several types of message body, which cover the majority of messaging styles currently in use.

All message types can be created using a `QueueSession` or a `TopicSession`. For more information on this refer to the sections "Using a `QueueSession`" and "Using a `TopicSession`" of this chapter.

## Message Bodies

The CRTL API defines five types of message body. They are:

- **Stream** A `StreamMessage` object's message body contains a stream of values of different datatypes. It is filled and read sequentially.
- **Map** A `MapMessage` object's message body contains a set of name-value pairs, where names are `mqstring` objects, and values are different C datatypes. The entries can be accessed sequentially or randomly by name. The order of the entries is undefined.
- **Text** A `TextMessage` object's message body contains an `mqstring` object. This message type can be used to transport plain-text messages, and XML messages.
- **Bytes** A `BytesMessage` object's message body contains a stream of uninterpreted bytes. This message type is for literally encoding a body to match an existing message format. In many cases, it is possible to use one of the other body types, which are easier to use. Although the JMS API allows the use of message properties with byte messages, they are typically not used, since the inclusion of properties may affect the format.

## Message Header

The `MessageHeader` contains standard information relating to a `Message` that is used by the FioranoMQ server and by the client applications. The header contains the following fields:

- `JMSType`
- `JMSMessageID`
- `JMSTimestamp`
- `JMSDeliveryMode`
- `JMSPriority`

- JMSExpiration
- JMSDestination
- JMSReplyTo
- JMSCorrelationID
- JMSRedelivered

The JMSMessageID and JMSTimestamp are set by the FioranoMQ server after a message send or publish call returns. Any values set by the client applications for these header fields are ignored.

The JMSDestination, JMSDeliveryMode, JMSPriority, and JMSExpiration are set to the values specified in the send or publish call after the call returns. Any values set by the client applications for these variables in the Message structure are ignored during the send/publish call.

The JMSCorrelationID header field is used for linking one message with another. It typically links a reply message with its requesting message. The following APIs are used to control the message header fields.

API	Function
MSG_setJMSCorrelationID	Sets the correlation ID for this message.
MSG_setJMSCorrelationIDAsBytes	Sets the correlation ID for this message as mqbyteArray
MSG_setJMSDelivery Mode	Sets the delivery mode for this message.
MSG_setJMSDestination	Sets the destination for this message.
MSG_setJMSExpiration	Sets the expiration value of the message.
MSG_setJMSMessageID	Sets the message ID for this message
MSG_setJMSPriority	Sets the priority for this message.
MSG_setJMSRedelivered	Set to indicate whether this message is being redelivered
MSG_setJMSReplyTo	Sets the destination, where a reply to this message should be sent
MSG_setJMSTimestamp	Sets the message timestamp as mqlong.
MSG_setJMSType	Sets the message type.
MSG_getJMSCorrelationID	Gets the correlation ID of the message as mqstring.
MSG_getJMSCorrelationIDAsBytes	Gets the correlation ID of the message as mqbyteArray
MSG_getJMSDeliveryMode	Gets the delivery mode for this message.
MSG_getJMSDestination	Gets the destination for this message.
MSG_getJMSExpiration	Gets the expiration time of the message.
MSG_getJMSMessageID	Gets the message ID for this message
MSG_getJMSPriority	Gets the message priority
MSG_getJMSRedelivered	Gets an indication of redelivery of this message



API	Function
MSG_getJMSReplyTo	MSG_getJMSReplyTo which a reply to this message should be sent
MSG_getJMSTimestamp	Gets the message timestamp for this message
MSG_getJMSType	Gets the message type

## Message Properties

A Message object contains a built-in facility for supporting application-defined property values. In effect, this provides a mechanism for adding application-specific header fields to a message.

Properties allow an application, via message selectors, to have the FioranoMQ server select, or filter, messages on its behalf using application-specific criteria.

Property names must obey the rules for a message selector identifier. Property names must not be null, and must not be empty strings. If a property name is set and it is either null or an empty string, an exception is thrown.

Property values can be mqboolean, mqbyte, mqshort, mqint, mqlong, mqfloat, mqdouble, and mqstring.

Property values are set prior to sending a message. When a client receives a message, its properties are in read-only mode. If a client attempts to set properties at this point, an exception is thrown. If clearProperties is called, the properties can now be both read from and written to. Note that header fields are distinct from properties. Header fields are never in read-only mode.

A property value may duplicate a value in a message's body, or it may not. For best performance, applications should use message properties only when they need to customize a message's header. The primary reason for doing this is to support customized message selection.

The following APIs can be used to set and get different properties in a Message structure.

API	Function
MSG_setBooleanProperty	Sets an mqboolean property value, into the Message, with the given name
MSG_setByteProperty	Sets an mqbyte property value, into the Message, , with the given name
MSG_setDoubleProperty	Sets an mqdouble property value, into the Message, with the given name
MSG_setFloatProperty	Sets an mqfloat property value, into the Message, with the given name
MSG_setIntProperty	Sets an mqint property value, into the Message, with the given name
MSG_setLongProperty	Sets an mqlong property value, into the Message, with the given name
MSG_setObjectProperty	Sets an Object as a property on a message
MSG_setShortProperty	Sets an mqshort property value, into the Message, with the given name
MSG_setStringProperty	Sets an mqstring property value, into the Message, with the given name

API	Function
MSG_getBooleanProperty	Gets the mqboolean property value with the given name
MSG_getByteProperty	Gets the mqbyte property value with the given name
MSG_getDoubleProperty	Gets the mqdouble property value with the given name
MSG_getFloatProperty	Gets the mqfloat property value with the given name
MSG_getIntProperty	Gets the mqint property value with the given name
MSG_getLongProperty	Gets the mqlong property value with the given name
MSG_getObjectProperty	Gets an Object Property from the message
MSG_getShortProperty	Gets the mqshort property value with the given name
MSG_getStringProperty	Gets the mqstring property value with the given name
MSG_getPropertyNames	Gets an array of all the property names
MSG_propertyExists	Checks if a property value exists
MSG_clearProperties	Clears the properties of the message. The message header fields and body are not cleared

## BytesMessage

A BytesMessage object is used to send a message containing a stream of uninterpreted bytes. The receiver of the message supplies the interpretation of the bytes.

This message type is for client encoding of existing message formats. If possible, one of the other self-defining message types should be used instead.

Although the JMS API allows the use of message properties with byte messages, they are typically not used. This is because the inclusion of properties may affect the format.

The data types can be written explicitly using methods for each type. When the message is first created, and when clearBody is called, the body of the message is in write-only mode. After the first call to reset has been made, the message body is in read-only mode. After a message has been sent, the client that sent it can retain and modify it without affecting the message that has been sent. The same message object can be sent multiple times. When a message has been received, the provider has called reset so that the message body is in read-only mode for the client.

If clearBody is called on a message in read-only mode, the message body is cleared and the message is in write-only mode. If a client attempts to read a message in write-only mode, an exception is thrown. If a client attempts to write a message in read-only mode, an exception is thrown.

Following APIs can be used to control the data in a bytes message.

API	Function
FBMSG_clearBody	Clears the body of the bytes message. s

<b>API</b>	<b>Function</b>
FBMSG_readBoolean	Reads an mqboolean from the bytes message stream.
FBMSG_readByte	Reads a signed 8-bit value from the bytes message stream.
FBMSG_readBytes	Reads an mqbyteArray of the specified length from the bytes message stream.
FBMSG_readChar	Reads a Unicode character value from the bytes message stream.
FBMSG_readDouble	Reads an mqdouble from the bytes message stream.
FBMSG_readFloat	Reads an mqfloat from the bytes message stream.
FBMSG_readInt	Reads a 32 bit signed integer (mqint) from the bytes message stream.
FBMSG_readLong	Reads a signed 64-bit integer (mqlong) from the bytes message stream.
FBMSG_readShort	Reads an mqshort from the bytes message stream.
FBMSG_readUnsignedByte	Reads an unsigned 8-bit number from the bytes message stream.
FBMSG_readUnsignedShort	Reads an unsigned 16-bit number (mqshort) from the bytes message stream
FBMSG_readUTF	Reads in mqstring that has been encoded using a modified UTF-8 format from the bytes message stream
FBMSG_reset	Puts the message body in read-only mode, and repositions the stream of bytes to the beginning
FBMSG_writeBoolean	Writes an mqboolean to the bytes message stream as a 1-byte value
FBMSG_writeByte	Writes an mqbyte to the bytes message stream as a 1-byte value
FBMSG_writeBytes	Write an mqbyteArray to the bytes message stream
FBMSG_writeBytesWithOffset	Writes a portion of mqbyteArray to the bytes message stream
FBMSG_writeChar	Writes an mqchar as 2 bytes to the bytes message stream
FBMSG_writeDouble	Writes an mqdouble to the bytes message stream.
FBMSG_writeFloat	Writes an mqfloat to the bytes message stream
FBMSG_writeInt	Writes an mqint to the bytes message stream.
FBMSG_writeLong	Writes an mqlong to the bytes message stream

API	Function
FBMSG_writeShort	Writes an mqshort to the bytes message stream
FBMSG_writeUTF	Write an mqstring to the bytes message stream, using UTF-8 encoding in a machine-independent manner

## MapMessage

A MapMessage object is used to send a set of name-value pairs. The names are mqstring objects, and the values are different data types of the C programming language. 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. The order of the entries is undefined.

The data-types can be read or written explicitly using methods for each type.

When a client receives a MapMessage, it is in read-only mode. If a client attempts to write to the message at this point, an exception is thrown. If clearBody is called, the message can now be both read from and written to.

API	Function
FMMSG_clearBody	Clears the contents of the message.
FMMSG_getBoolean	Gets the mqboolean value with the given name.
FMMSG_getByte	Gets the mqbyte value with the given name.
FMMSG_getBytes	Gets the mqbyteArray with the given name.
FMMSG_getChar	Gets the mqchar value with the given name.
FMMSG_getDouble	Gets the mqdouble value with the given name.
FMMSG_getFloat	Gets the mqfloat value with the given name.
FMMSG_getInt	Gets the mqint value with the given name.
FMMSG_getLong	Gets the mqlong value with the given name.
FMMSG_getMapNames	Gets an array of all the map names.
FMMSG_getShort	Gets the mqshort value with the given name.
FMMSG_getString	Gets the mqstring value with the given name.
FMMSG_itemExists	Checks if an item with the given name exists in this MapMessage
FMMSG_setBoolean	Sets the mqboolean value with the given name.
FMMSG_setByte	Sets the mqbyte value with the given name.
FMMSG_setBytes	Sets the mqbyteArray with the given name.
FMMSG_setBytesWithOffset	Sets a portion of mqbyteArray with the given name.
FMMSG_setChar	Sets the mqchar value with the given name.

API	Function
FMMSG_setDouble	Sets the mqdouble value with the given name.
FMMSG_setFloat	Sets the mqfloat value with the given name.
FMMSG_setInt	Sets the mqint value with the given name.
FMMSG_setLong	Sets the mqlong value with the given name.
FMMSG_setShort	Sets the mqshort value with the given name.
FMMSG_setString	Sets the mqstring value with the given name.

## StreamMessage

A StreamMessage object is used to send a stream of data types of the C programming language. It is filled and read sequentially.

The data-types can be read or written explicitly using methods for each type.

When the message is first created, and when clearBody is called, the body of the message is in write-only mode. After the first call to reset has been made, the message body is in read-only mode. After a message has been sent, the client that sent it can retain and modify it without affecting the message that has been sent. The same message object can be sent multiple times. When a message is received, the CRTL has called reset so that the message body is in read-only mode for the client.

- If clearBody is called on a message in read-only mode, the message body is cleared and the message body is in write-only mode.
- If a client attempts to read a message in write-only mode, an exception is thrown. If a client attempts to write a message in read-only mode, an exception is thrown.

API	Function
FSMSG_clearBody	Clears the contents of the stream message
FSMSG_readBoolean	Reads an mqboolean from the stream message
FSMSG_readByte	Reads an mqbyte from the stream message
FSMSG_readBytes	Reads an mqbyteArray from the stream message
FSMSG_readChar	Reads an mqchar from the stream message
FSMSG_readDouble	Reads an mqdouble from the stream message.
FSMSG_readFloat	Reads an mqfloat from the stream message
FSMSG_readInt	Reads an mqint from the stream message
FSMSG_readLong	Reads an mqlong from the stream message
FSMSG_readShort	Reads an mqshort from the stream message
FSMSG_readString	Reads an mqstring from the stream message
FSMSG_reset	Puts the message body in read-only mode, and repositions the stream to the beginning.
FSMSG_writeBoolean	Writes an mqboolean to the stream message, as a 1-byte

	value
FSMSG_writeByte	Writes an mqbyte to the stream message
FSMSG_writeBytes	Writes an mqbyteArray to the stream message
FSMSG_writeBytesWithOffset	Writes a portion of an mqbyteArray to the stream message
FSMSG_writeChar	Writes an mqchar to the stream message
FSMSG_writeDouble	Writes an mqdouble to the stream message
FSMSG_writeFloat	Writes an mqfloat to the stream message
FSMSG_writeInt	Writes an mqint to the stream message
FSMSG_writeLong	Writes an mqlong to the stream message
FSMSG_writeShort	Writes an mqshort to the stream message
FSMSG_writeString	Writes an mqstring to the stream message

## TextMessage

A TextMessage object is used to send a message containing an mqstring. It inherits from the Message interface and adds a text message body.

This message type can be used to transport text-based messages, including those with XML content.

When a client receives a TextMessage, it is in read-only mode. If a client attempts to write to the message at this point, an exception is thrown. If clearBody is called, the message can now be both read from and written to.

API	Function
FTMSG_getText	Gets the string containing data of this message
FTMSG_setText	Sets the string containing data of this message.

## Acknowledging a message

The CRTL provides the acknowledge API in the Message which is equivalent to the Message.acknowledge API of JMS. A call to this API acknowledges all messages delivered in that session till the message on which the call is made.

This call is relevant for sessions that are created using the CLIENT\_ACKNOWLEDGEMENT mode. In order to rollback messages received on a client ack session the TS\_recover API can be used.

API	Function
MSG_acknowledge	Acknowledges all consumed messages of the session of this

	consumed message.
--	-------------------

## Destroying a message

A Message structure should be destroyed by the Client application after the message has been used to avoid memory leaks.

All types of messages can be destroyed using the following API call.

API	Function
MSG_free	Frees the resources allocated for the message

## Large Message Support

With Large Message Support (LMS) in FioranoMQ, clients can transfer large files in the form of large messages with theoretically no limit on the message size. Large messages can be attached with any JMS message and the client can be sure of a reliable and secure transfer of the message through FioranoMQ server. Please refer to chapter 15 of **FioranoMQ concepts guide** for detailed explanation of LMS.

The following functions can be used to use LMS implementation.

### LMSG\_getMessageStatus

**Declaration:** LMTransferStatus LMSG\_getMessageStatus(Message msg);

Gets the status of the message. Status includes number of bytes transferred, number of bytes left to be transferred, and so on.

### LMSG\_setLMStatusListener

**Declaration:** void LMS\_LMSG\_setLMStatusListener(Message msg , LMStatusListener listener, int updateFrequency);

Sets the status listener for the message. This function is used to know the status of message transfer asynchronously.

LMStatusListener is defined as

```
typedef void (*LMStatusListener)(LMTransferStatus status, mqboolean exOccurred);
```

### LMSG\_getLMStatusListener

**Declaration:** LMStatusListener LMSG\_getLMStatusListener(Message msg);

Gets the status listener for the message.

## LMSG\_saveTo

**Declaration:** void LMSG\_saveTo(Message msg, mqstring fileName, mqboolean isBlocking);

Saves the contents of the message in the file specified.

## LMSG\_resumeSaveTo

**Declaration:** void LMSG\_resumeSaveTo(Message msg, mqboolean isBlocking);

Resumes saving the contents of the message in the file specified.

## LMSG\_resumeSend

**Declaration:** void LMSG\_resumeSend(Message msg);

Resumes an incomplete transfer. This function is used to resume a message transfer which could not be completed earlier either due to some internal error or due to some problem at the client side.

## LMSG\_cancelAllTransfers

**Declaration:** void LMSG\_cancelAllTransfers(Message msg);

Cancels all message transfers which are currently transferring this message. A cancelled transfer also removes the resume information of the transfer. Hence a transfer once cancelled cannot be resumed.

## LMSG\_cancelTransfer

**Declaration:** void LMSG\_cancelTransfer(Message msg, int consumerID);

Cancels the transfer specified by the consumerID.

A cancelled transfer also removes the resume information of the transfer. Hence a transfer once cancelled cannot be resumed.

## LMSG\_suspendAllTransfers

**Declaration:** void LMSG\_suspendAllTransfers(Message msg);

Suspends all the message transfers which are transferring this large message temporarily.

Suspending a transfer only stops the thread which is doing the message transfer and does not delete resume related information. Hence, a suspended transfer can be resumed using resume functions.



### LMSG\_suspendTransfer

**Declaration:** void LMSG\_suspendTransfer(Message msg, int consumerID);

Suspends the transfer specified by the consumerID temporarily.

Suspending a transfer only stops the thread which is doing the message transfer and does not delete resume related information. Hence, a stopped transfer can be resumed using resume functions.

### LMSG\_setFragmentSize

**Declaration:** void LMSG\_setFragmentSize(Message msg, int size);

Sets the fragment size for the message.

### LMSG\_getFragmentSize

**Declaration:** int LMSG\_getFragmentSize(Message msg);

gets the fragment size of the message.

### LMSG\_setWindowSize

**Declaration:** void LMSG\_setWindowSize(Message msg, int size);

Sets the frequency after which acknowledgement will be sent

### LMSG\_getWindowSize

**Declaration:** int LMSG\_getWindowSize(Message msg);

gets the window size of the message.

### LMSG\_setRequestTimeoutInterval

**Declaration:** void LMSG\_setRequestTimeoutInterval(Message msg, long timeout);

Sets the time until which the sender will wait for message transfer to start

### LMSG\_getRequestTimeoutInterval

**Declaration:** long LMSG\_getRequestTimeoutInterval(Message msg);

Gets the time until which the sender will wait for message transfer to start

### LMSG\_setResponseTimeoutInterval

**Declaration:** void LMSG\_setResponseTimeoutInterval(Message msg, long responseInterval);

Sets the time until which the sender/receiver will wait for message from the other end.

### LMSG\_getResponseTimeoutInterval

**Declaration:** long LMSG\_getResponseTimeoutInterval(Message msg);

Gets the time until which the sender/receiver will wait for messages from the other end. The following functions are to get the status information from LMTransferStatus Structure.

### LMTS\_getBytesTransferred

**Declaration:** mqlong LMTS\_getBytesTransferred(LMTransferStatus lmts);

Returns the number of bytes transferred.

### LMTS\_getBytesToTransfer

**Declaration:** mqlong LMTS\_getBytesToTransfer(LMTransferStatus lmts);

Returns the number of bytes to be transferred.

### LMTS\_getLastFragmentID

**Declaration:** mqlong LMTS\_getLastFragmentID(LMTransferStatus lmts);

Returns the ID of the last fragment.

### LMTS\_getPercentageProgress

**Declaration:** float LMTS\_getPercentageProgress(LMTransferStatus lmts);

Returns the percentage of the progress of message transfer.

### LMTS\_getStatus

**Declaration:** mqlbyte LMTS\_getStatus(LMTransferStatus lmts);

Returns the status of the Message transfer.

### LM\_TRANSFER\_NOT\_INIT or 1

Indicates that the transfer has not yet started

### LM\_TRANSFER\_IN\_PROGRESS or 2

Indicates that transfer is currently in progress

### LM\_TRANSFER\_DONE or 3

Indicates that the transfer is complete for one consumer

### LM\_TRANSFER\_ERR or 4

Indicates that an error occurred during the transfer

### LM\_ALL\_TRANSFER\_DONE or 5

Indicates that the transfer is complete for all consumers

### LMTS\_isTransferComplete

**Declaration:** mqboolean LMTS\_isTransferComplete(LMTransferStatus lmts);

Returns TRUE if transfer completes, else returns FALSE.

### LMTS\_isTransferCompleteForAll

**Declaration:** mqboolean LMTS\_isTransferCompleteForAll(LMTransferStatus lmts);

Returns TRUE if all the transfers are completed, else returns FALSE.

### LMTS\_getLargeMessage

**Declaration:** Message LMTS\_getLargeMessage(LMTransferStatus lmts);

Returns the large message for which this status is created.

### LMTS\_getConsumerID

**Declaration:** int LMTS\_getConsumerID(LMTransferStatus lmts);

Returns the consumerID of the connection that is being used.

These functions are used to get any pending large messages.

### LMC\_getUnfinishedMessagesToSend

**Declaration:** RMEEnum LMC\_getUnfinishedMessagesToSend(FioranoConnection fc);

Gets the enumeration of Messages whose transfers are pending as they were not fully sent.

### LMC\_getUnfinishedMessagesToReceive

**Declaration:** RMEEnum LMC\_getUnfinishedMessagesToReceive(FioranoConnection fc);

Gets the enumeration of Messages which were not fully received and need to be resumed.

RMEEnum is Recoverable Messages Enumeration. The following functions should be called to use the RMEEnum structure.

### RME\_hasMoreElements

**Declaration:** mqboolean RME\_hasMoreElements(RMEEnum enumr);

Returns true if there are more elements in the enumeration, false otherwise.

### RME\_nextElement

**Declaration:** mqobject RME\_nextElement(RMEEnum enumr);

Return the next element (Message Object) of the enumeration.

## Error Handling

The error handling mechanism in CRTL is very similar to Exception handling in Java. An exception/error stack is maintained and the errors occurred during a call to any API are pushed into this stack automatically. When the call returns, the user should check for any error occurred during that call and take necessary steps. For this purpose, the user is provided with the following functions, which operate on this error stack.

API	Function
MqExceptionOccurred()	Returns a Boolean value indicating whether an exception occurred, in the most recent API call.
MqPrintException()	Prints the exception stack, including the error code and a description
MqExceptionClear()	Clears the stack of exceptions occurred until this call
MqGetLastErrCode()	Gets the error code of the last exception pushed in the exception stack

The MqExceptionOccurred() can be avoided by checking the return value of the API call.

- The APIs, which don't return any value in Java, are declared to return an mqboolean in CRTL indicating the success/failure of the call.
- The APIs, which return basic data types in Java, are declared to return an mqboolean and take a value-result argument of that type in CRTL. The mqboolean value returned indicates the success of the call

- The APIs, which return an object in Java, return a pointer in CRTL. The return value is NULL if an error occurred during the call.

The MqExceptionClear() function should be called after every failed API call. Or, it is possible that the following API calls fail due to the exception that occurred during this call.

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 MqGetLastErrCode() should be used, when the user wants to check what error has occurred in the API call and proceed accordingly. On the other hand, the MqPrintException() is used to print a detailed description of the Error, which leads to the root of the exception.

## Utility APIs

### Key-value pairs in Hashtable

Hashtable represents a pointer to an internally maintained structure, which maintains key value pairs.

API	Function
HT_Clear	Clears this hashtable so that it contains no keys and/or elements.
HT_ContainsKey	Tests if the specified mqstring is a key in this hash-table.
HT_ContainsValue	Returns true if this hashtable maps one or more keys to this value.
HT_Get	Returns the value to which the specified key is mapped in this hashtable.
HT_IsEmpty	Tests if this hashtable maps no keys to values.
HT_Put	Maps the specified key to the specified value in the argument hashtable.
HT_RemoveElement	Removes the key (and its corresponding value) from this hashtable.
HT_Size	Returns the number of keys in this hashtable.

# Chapter 4: Function Reference

---

## getRuntimeVersion

### Purpose

Returns a string containing version of CRTL and date on which it is built.

### Declaration

```
mqstring getRuntimeVersion();
```

### Include

Utils.h

## DestroyHashtable

### Purpose

Cleans the hashtable structure.

### Declaration

```
void DestroyHashtable(Hashtable table);
```

### Include

fiorano\_hashtable.h

### Parameters

Table

The hashtable that has to be destroyed

## ES\_free

### Purpose

Frees all the memory associated with the exception stack trace.

### Declaration

```
void ES_free();
```

### Include

mq\_errno.h

## FBMSG\_clearBody

### Purpose

Clears the body of the bytes message. This does not include the message header and properties.

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

```
mqboolean FBMSG_clearBody(BytesMessage msg);
```

### Include

```
bytes_message.h
```

### Parameters

Msg

BytesMessage

### Return Values

Mqboolean

TRUE if successful, FALSE if an error occurs

## FBMSG\_readBoolean

### Purpose

Reads an mqboolean from the bytes message stream.

### Declaration

```
mqboolean FBMSG_readBoolean(BytesMessage msg, mqboolean* value);
```

### Include

```
bytes_message.h
```

### Parameters

Msg

BytesMessage

value

Pointer to the mqboolean into which data is to be read

**Return Value**

Mqboolean

TRUE if successful, FALSE if error occurs

**FBMSG\_readByte****Purpose**

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

**Declaration**

```
mqboolean FBMSG_readByte(BytesMessage msg, mqbyte* value);
```

**Include**

bytes\_message.h

**Parameters**

Msg

BytesMessage

Value

Pointer to the mqbyte, into which the next byte from the bytes message stream is read as a signed 8-bit byte.

**Return Values**

Mqboolean

TRUE if successful, FALSE if error occurs

**FBMSG\_readBytes****Purpose**

Reads an mqbyteArray of the specified length 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 is 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.

If length is negative, or length is greater than the length of the array value, then an OutOfBounds error is reported. For this exception case, no bytes are read from the stream.



```
mqint FBMSG_readBytes(BytesMessage msg, mqbyteArray value, mqint length);
```

**Include**

bytes\_message.h

**Parameters**

Msg

BytesMessage

Value

mqbyteArray into which the data is to be read

length

Number of bytes to be read into "value", hence must be less than or equal to the length of "value"

**Returns Value**

Mqint

Number of bytes read, -1 if an error occurs

## FBMSG\_readChar

**Purpose**

Reads a Unicode character value from the bytes message stream

**Declaration**

```
mqboolean FBMSG_readChar(BytesMessage msg, mqchar* value);
```

**Include**

bytes\_message.h

**Parameters**

Msg

BytesMessage

Value

Pointer to the mqchar, into which the next two bytes (java char) are read.

Mqboolean

TRUE if successful, FALSE if error occurs

## FBMSG\_readDouble

### Purpose

Reads an mqdouble from the bytes message stream.

### Declaration

```
mqboolean FBMSG_readDouble(BytesMessage msg, mqdouble* value);
```

### Include

bytes\_message.h

### Parameters

msg

BytesMessage

Value

Pointer to the mqdouble, into which the next eight bytes of the bytes message stream are read and interpreted as mqdouble.

### Return Values

mqboolean

TRUE if successful, FALSE if error occurs

## FBMSG\_readFloat

### Purpose

Reads an mqfloat from the bytes message stream.

### Declaration

```
mqboolean FBMSG_readFloat(BytesMessage msg, mqfloat* value);
```

### Include

bytes\_message.h

### Parameters

msg

BytesMessage

Value

Pointer to the mqfloat, into which the next four bytes of the bytes message stream are read and interpreted as mqfloat.

## Return Values

Mqboolean

TRUE if successful, FALSE if error occurs

## FBMSG\_readInt

### Purpose

Reads a 32 bit signed integer (mqint) from the bytes message stream.

### Declaration

```
mqboolean FBMSG_readInt(BytesMessage msg, mqint* value);
```

### Include

bytes\_message.h

### Parameters

Msg

BytesMessage

Value

Pointer to the mqint, into which the next four bytes of the bytes message stream are read and interpreted as mqint.

### Return Values

Mqboolean

TRUE if successful, FALSE if error occurs

## FBMSG\_readLong

### Purpose

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

### Declaration

```
mqboolean FBMSG_readLong(BytesMessage msg, mqlong* value);
```

### Include

bytes\_message.h

### Parameters

Msg

BytesMessage

Value

Pointer to mqlong, into which the next eight bytes are read from the bytes message stream, interpreted as mqlong.

### Return Values

Mqboolean

TRUE if successful, FALSE if error occurs

## FBMSG\_readShort

### Purpose

Reads an mqshort from the bytes message stream.

### Declaration

```
mqboolean FBMSG_readShort(BytesMessage msg, mqshort* value);
```

### Include

bytes\_message.h

### Parameters

Msg

BytesMessage

Value

Pointer to mqshort, into which the next two bytes from the bytes message stream, interpreted as a signed 16-bit number, are read.

### Return Values

Mqboolean

TRUE if successful, FALSE if error occurs

## FBMSG\_readUnsignedByte

### Purpose

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

### Declaration

```
mqboolean FBMSG_readUnsignedByte(BytesMessage msg, mqint* value);
```

### Include

bytes\_message.h

### Parameters

Msg

BytesMessage

Value

Pointer to the mqint, into which the next byte from the bytes message stream is read as an unsigned 8-bit number.

### Return Values

Mqboolean

TRUE if successful, FALSE if error occurs

## FBMSG\_readUnsignedShort

### Purpose

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

### Declaration

```
mqboolean FBMSG_readUnsignedShort(BytesMessage msg, mqint* value);
```

### Include

bytes\_message.h

### Parameters

Msg

BytesMessage

Value

**Return Values**

Mqboolean

TRUE if successful, FALSE if error occurs

**FBMSG\_readUTF****Purpose**

Reads in mqstring that has been encoded using a modified UTF-8 format from the bytes message stream.

**Declaration**

```
mqstring FBMSG_readUTF(BytesMessage msg);
```

**Include**

bytes\_message.h

**Parameters**

Msg

BytesMessage

**Return Values**

Mqstring

mqstring read from the bytes message, NULL if an error occurs

**FBMSG\_reset****Purpose**

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

**Declaration**

```
mqboolean FBMSG_reset(BytesMessage msg);
```

**Include**

bytes\_message.h

**Parameters**

msg

BytesMessage

**Return Values**

Mqboolean

TRUE if successful, FALSE if error occurs

**FBMSG\_writeBoolean****Purpose**

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

**Declaration**

```
mqboolean FBMSG_writeBoolean(BytesMessage msg, mqboolean value);
```

**Include**

bytes\_message.h

**Parameters**

Msg

BytesMessage

Value

mqboolean value to be written to the bytes message

**Return Values**

Mqboolean

TRUE if successful, FALSE if error occurs

**FBMSG\_writeByte****Purpose**

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

**Declaration**

```
mqboolean FBMSG_writeByte(BytesMessage msg, mqbyte value);
```

**Include**

bytes\_message.h

### Parameters

Msg

BytesMessage

Value

mqbyte value to be written to the bytes message

### Return Values

Mqboolean

TRUE if successful, FALSE if error occurs

## FBMSG\_writeBytes

### Purpose

Write an mqbyteArray to the bytes message stream.

### Declaration

```
mqboolean FBMSG_writeBytes(BytesMessage msg, mqbyteArray value,  
mqint size);
```

### Include

bytes\_message.h

### Parameters

Msg

BytesMessage

Value

mqbyteArray to be written to the bytes message  
size

Size of the mqbyteArray

### Return Values

Mqboolean

TRUE if successful, FALSE if error occurs



## FBMSG\_writeBytesWithOffset

### Purpose

Writes a portion of mqbyteArray to the bytes message stream.

### Declaration

```
mqboolean FBMSG_writeBytesWithOffset(BytesMessage msg, mqbyteArray value, mqint  
offset, mqint length);
```

### Include

bytes\_message.h

### Parameters

Msg

BytesMessage

Value

mqbyteArray to be written to the bytes message

offset

Initial offset within the mqbyteArray

Length

Number of bytes to use

### Return Values

Mqboolean

TRUE if successful, FALSE if error occurs

## FBMSG\_writeChar

### Purpose

Writes an mqchar as 2 bytes to the bytes message stream.

### Declaration

```
mqboolean FBMSG_writeChar(BytesMessage msg, mqchar value);
```

### Include

bytes\_message.h

### Parameters

msg

BytesMessage

Value

mqchar value to be written to the bytes message

### Return Values

Mqboolean

TRUE if successful, FALSE if error occurs

## FBMSG\_writeDouble

### Purpose

Writes an mqdouble to the bytes message stream.

### Declaration

```
mqboolean FBMSG_writeDouble(BytesMessage msg, mqdouble value);
```

### Include

bytes\_message.h

### Parameters

Msg

BytesMessage

Value

mqdouble value to be written to the bytes message

### Return Values

Mqboolean

TRUE if successful, FALSE if error occurs

## FBMSG\_writeFloat

### Purpose

Writes an mqfloat to the bytes message stream.

### **Declaration**

```
mqboolean FBMSG_writeFloat(BytesMessage msg, mqfloat value);
```

### **Include**

bytes\_message.h

### **Parameters**

Msg

BytesMessage

Value

mqfloat value to be written to the bytes message

### **Return Values**

Mqboolean

TRUE if successful, FALSE if error occurs

## **FBMSG\_writeInt**

### **Purpose**

Writes an mqint to the bytes message stream.

### **Declaration**

```
mqboolean FBMSG_writeInt(BytesMessage msg, mqint value);
```

### **Include**

bytes\_message.h

### **Parameters**

Msg

BytesMessage

Value

mqint value to be written to the bytes message

### **Return Values**

Mqboolean

TRUE if successful, FALSE if error occurs

## FBMSG\_writeLong

### Purpose

Writes an mqlong to the bytes message stream.

### Declaration

```
mqboolean FBMSG_writeLong(BytesMessage msg, mqlong value);
```

### Include

bytes\_message.h

### Parameters

Msg

BytesMessage

Value

mqlong value to be written to the bytes message

### Return Values

Mqboolean

TRUE if successful, FALSE if error occurs

## FBMSG\_writeShort

### Purpose

Writes an mqshort to the bytes message stream.

### Declaration

```
mqboolean FBMSG_writeShort(BytesMessage msg, mqshort value);
```

### Include

bytes\_message.h

### Parameters

Msg

BytesMessage

Value

mqshort value to be written to the bytes message

## Return Values

mqboolean

TRUE if successful, FALSE if error occurs

## FBMSG\_writeUTF

### Purpose

Write an mqstring to the bytes message stream, using UTF-8 encoding in a machine-independent manner.

### Declaration

```
mqboolean FBMSG_writeUTF(BytesMessage msg, mqstring value);
```

### Include

bytes\_message.h

### Parameters

Msg

BytesMessage

Value

mqstring value to be written to the bytes message

### Return Values

Mqboolean

TRUE if successful, FALSE if error occurs

## FMMSG\_clearBody

### Purpose

Clears the contents of the message.

### Declaration

```
mqboolean FMMSG_clearBody(MapMessage msg);
```

### Include

map\_message.h

### Parameters

msg

MapMessage

### Return Values

Mqboolean

TRUE if successful, FALSE if an error occurs

## FMMSG\_getBoolean

### Purpose

Gets the mqboolean value with the given name.

### Declaration

```
mqboolean FMMSG_getBoolean(MapMessage msg, mqstring name, mqboolean* value);
```

### Include

map\_message.h

### Parameters

Msg

MapMessage

Name

Name of the mqboolean

Value

Pointer to the mqboolean, into which data is to be read

### Return Values

Mqboolean

TRUE if successful, FALSE if error occurs

## FMMSG\_getByte

### Purpose

Gets the mqbyte value with the given name.

### Declaration

```
mqboolean FMMSG_getByte(MapMessage msg, mqstring name, mqbyte* value);
```

### Include

map\_message.h

### Parameters

Msg

MapMessage

name

Name of the mqbyte

Value

Pointer to the mqbyte, into which data is to be read

### Return Values

Mqboolean

TRUE if successful, FALSE if error occurs

## FMMSG\_getBytes

### Purpose

Gets the mqbyteArray with the given name.

### Declaration

```
mqint FMMSG_getBytes(MapMessage msg, mqstring name, mqbyteArray bArray, mqint length);
```

### Include

map\_message.h

### Parameters

Msg

MapMessage

Name

Name of the mqbyteArray

bArray

value

Size of the specified mqbyteArray

### **Return Values**

Mqint

Number of bytes read, -1 if no byte is read

## **FMMSG\_getChar**

### **Purpose**

Gets the mqchar value with the given name.

### **Declaration**

```
mqboolean FMMSG_getChar(MapMessage msg, mqstring name, mqchar* value);
```

### **Include**

map\_message.h

### **Parameters**

Msg

MapMessage

Name

Name of the mqchar

Value

Pointer to the mqchar, into which data is to be read

### **Return Values**

Mqboolean

TRUE if successful, FALSE if an error occurs

## **FMMSG\_getDouble**

### **Purpose**

Gets the mqdouble value with the given name.



### **Declaration**

```
mqboolean FMMSG_getDouble(MapMessage msg, mqstring name, mqdouble* value);
```

### **Include**

map\_message.h

### **Parameters**

Msg

MapMessage

Name

Name of the mqdouble

Value

Pointer to the mqdouble, into which data is to be read

### **Return Values**

Mqboolean

TRUE if successful, FALSE if an error occurs

## **FMMSG\_getFloat**

### **Purpose**

Gets the mqfloat value with the given name.

### **Declaration**

```
mqboolean FMMSG_getFloat(MapMessage msg, mqstring name, mqfloat* value);
```

### **Include**

map\_message.h

### **Parameters**

Msg

MapMessage

Name

Name of the mqfloat

Value

Pointer to the mqfloat, into which data is to be read

## Return Values

Mqboolean

TRUE if successful, FALSE if an error occurs

## FMMSG\_getInt

### Purpose

Gets the mqint value with the given name.

### Declaration

```
mqboolean FMMSG_getInt(MapMessage msg, mqstring name, mqint*  
value);
```

### Include

map\_message.h

### Parameters

Msg

MapMessage

Name

Name of the mqint

Value

Pointer to the mqint, into which data is to be read

### Return Values

Mqboolean

TRUE if successful, FALSE if an error occurs

## FMMSG\_getLong

### Purpose

Gets the mqlong value with the given name.

### Declaration

```
mqboolean FMMSG_getLong(MapMessage msg, mqstring name, mqlong*  
value);
```

### **Include**

map\_message.h

### **Parameters**

Msg

MapMessage

Name

Name of the mqlong

Value

Pointer to the mqlong, into which data is to be read

### **Return Values**

Mqboolean

TRUE if successful, FALSE if error occurs

## **FMMSG\_getMapNames**

### **Purpose**

Gets an array of all the map names.

### **Declaration**

```
mqint FMMSG_getMapNames(MapMessage msg, mqstring** namesPointer);
```

### **Include**

map\_message.h

### **Parameters**

Msg

MapMessage

namesPointer

Pointer to an array of strings, into which the map names are set

### **Return Values**

Mqint

Number of map names, -1 if an error occurs

## FMMSG\_getShort

### Purpose

Gets the mqshort value with the given name.

### Declaration

```
mqboolean FMMSG_getShort(MapMessage msg, mqstring name, mqshort* value);
```

### Include

map\_message.h

### Parameters

Msg

MapMessage

Name

Name of the mqshort

Value

Pointer to the mqshort, into which data is to be read

### Return Values

Mqboolean

TRUE if successful, FALSE if an error occurs

## FMMSG\_getString

### Purpose

Gets the mqstring value with the given name.

### Declaration

```
mqstring FMMSG_getString(MapMessage msg, mqstring name);
```

### Include

map\_message.h

### Parameters

Msg

MapMessage Name of the mqstring

### **Return Values**

Mqstring

mqstring with the give name, NULL if there is no item by this name

## **FMMSG\_itemExists**

### **Purpose**

Checks if an item with the given name exists in this MapMessage.

### **Declaration**

```
mqboolean FMMSG_itemExists(MapMessage msg, mqstring name);
```

### **Include**

map\_message.h

### **Parameters**

Msg

MapMessage

Name

Name of the item

### **Return Values**

Mqboolean

TRUE if the value exists; false otherwise

## **FMMSG\_setBoolean**

### **Purpose**

Sets the mqboolean value with the given name.

### **Declaration**

```
mqboolean FMMSG_setBoolean(MapMessage msg, mqstring name, mqboolean value);
```

### **Include**

map\_message.h

### **Parameters**

Msg

MapMessage

Name

Name of the mqboolean

Value

mqboolean value to be set in the map

### **Return Values**

Mqboolean

TRUE if successful, FALSE if an error occurs

## **FMMSG\_setByte**

### **Purpose**

Sets the mqbyte value with the given name.

### **Declaration**

```
mqboolean FMMSG_setByte(MapMessage msg, mqstring name, mqbyte value);
```

### **Include**

map\_message.h

### **Parameters**

Msg

MapMessage

Name

Name of the mqbyte

Value

mqbyte value to be set in the map

### **Return Values**

mqboolean

## FMMSG\_setBytes

### Purpose

Sets the mqbyteArray with the given name.

### Declaration

```
mqboolean FMMSG_setBytes(MapMessage msg, mqstring name, mqbyteArray value, mqint length);
```

### Include

map\_message.h

### Parameters

Msg

MapMessage

Name

Name of the mqboolean

Value

mqbyteArray value to be set in the map

length

Size of the specified mqbyteArray

### Return Values

Mqboolean

TRUE if successful, FALSE if an error occurs

## FMMSG\_setBytesWithOffset

### Purpose

Sets a portion of mqbyteArray with the given name.

### Declaration

```
mqboolean FMMSG_setBytesWithOffset(MapMessage msg, mqstring name, mqbyteArray value, mqint offset, mqint length);
```

### Includes

map\_message.h

### Parameters

Msg

MapMessage

Name

Name of the mqboolean

Value

mqbyteArray value to be set in the map

offset

Initial offset within the mqbyteArray

Length

Number of bytes to be used

### Return Values

Mqboolean

TRUE if successful, FALSE if an error occurs

## FMMSG\_setChar

### Purpose

Sets the mqchar value with the given name.

### Declaration

```
mqboolean FMMSG_setChar(MapMessage msg, mqstring name, mqchar value);
```

### Include

map\_message.h

### Parameters

Msg

MapMessage

Name

Name of the mqchar

Value

mqchar value to be set in the map



## Return Values

Mqboolean

TRUE if successful, FALSE if an error occurs

## FMMSG\_setDouble

### Purpose

Sets the mqdouble value with the given name.

### Declaration

```
mqboolean FMMSG_setDouble(MapMessage msg, mqstring name, mqdouble value);
```

### Include

map\_message.h

### Parameters

Msg

MapMessage

Name

Name of the mqdouble

Value

mqdouble value to be set in the map

### Return Values

Mqboolean

TRUE if successful, FALSE if an error occurs

## FMMSG\_setFloat

### Purpose

Sets the mqfloat value with the given name.

### Declaration

```
mqboolean FMMSG_setFloat(MapMessage msg, mqstring name, mqfloat value);
```

### Include

map\_message.h

### Parameters

Msg

MapMessage

Name

Name of the mqfloat

Value

mqfloat value to be set in the map

### Return Values

Mqboolean

TRUE if successful, FALSE if an error occurs

## FMMSG\_setInt

### Purpose

Sets the mqint value with the given name.

### Declaration

```
mqboolean FMMSG_setInt(MapMessage msg, mqstring name, mqint value);
```

### Include

map\_message.h

### Parameters

Msg

MapMessage

Name

Name of the mqint

Value

mqint value to be set in the map

### Return Values

Mqboolean

TRUE if successful, FALSE if an error occurs

## FMMSG\_setLong

### Purpose

Sets the mqlong value with the given name.

### Declaration

```
mqboolean FMMSG_setLong(MapMessage msg, mqstring name, mqlongvalue);
```

### Include

map\_message.h

### Parameters

Msg

MapMessage

Name

Name of the mqlong

Value

mqlong value to be set in the map

### Return Values

Mqboolean

TRUE if successful, FALSE if an error occurs

## FMMSG\_setShort

### Purpose

Sets the mqshort value with the given name.

### Declaration

```
mqboolean FMMSG_setShort(MapMessage msg, mqstring name, mqshort value);
```

### Include

map\_message.h

### Parameters

Msg

MapMessage

name

Name of the mqshort

Value

mqshort value to be set in the map

### **Return Values**

Mqboolean

TRUE if successful, FALSE if an error occurs

## **FMMSG\_setString**

### **Purpose**

Sets the mqstring value with the given name.

### **Declaration**

```
mqboolean FMMSG_setString(MapMessage msg, mqstring name, mqstringvalue);
```

### **Include**

map\_message.h

### **Parameters**

Msg

MapMessage

Name

Name of the mqstring

Value

mqstring value to be set in the map

### **Return Values**

Mqboolean

TRUE if successful, FALSE if an error occurs

## FSMSG\_clearBody

### Purpose

Clears the contents of the stream message.

### Declaration

```
mqboolean FSMSG_clearBody(StreamMessage msg);
```

### Include

```
stream_message.h
```

### Parameters

Msg

StreamMessage that is to be cleared.

### Return Values

Mqboolean

TRUE if successful, FALSE if an error occurs

## FSMSG\_readBoolean

### Purpose

Reads an mqboolean from the stream message.

### Declaration

```
mqboolean FSMSG_readBoolean(StreamMessage msg, mqboolean *value);
```

### Include

```
stream_message.h
```

### Parameters

Msg

StreamMessage

Value

Pointer to the mqboolean, into which data is read

**Return Values**

Mqboolean

TRUE if successful, FALSE if an error occurs

## FSMSG\_readByte

**Purpose**

Reads an mqbyte from the stream message.

**Declaration**

```
mqboolean FSMSG_readByte(StreamMessage msg, mqbyte *value);
```

**Include**

stream\_message.h

**Parameters**

Msg

StreamMessage

Value

Pointer to the mqbyte, into which the next byte from the stream message (as a 8bit byte) is read.

**Return Values**

Mqboolean

TRUE if successful, FALSE if an error occurs

## FSMSG\_readBytes

**Purpose**

Reads an mqbyteArray from the stream message.

To read the field value, readBytes should be successively called until it returns a value less than the length of the read buffer. The value of the bytes in the buffer following the last byte read is undefined.

If readBytes returns a value equal to the length of the buffer, a subsequent read Bytes call must be made. If there are no more bytes to be read, this call returns -1.

If the byte array field value is null, readBytes returns -1. If the byte array field value is empty, readBytes returns 0.

Once the first `readBytes` call on a `byte[]` field value has been made, the full value of the field must be read before it is valid to read the next field. An attempt to read the next field before that has been done leads to an error.

**Declaration**

```
mqint FSMSG_readBytes(StreamMessage msg, mqbyteArray value, mqint length);
```

**Include**

```
stream_message.h
```

**Parameters**

Msg

StreamMessage

Value

mqbyteArray into which the data is read

length

Size of the mqbyteArray

**Return Values**

Mqint

Number of bytes read

## FSMSG\_readChar

**Purpose**

Reads an `mqchar` from the stream message.

**Declaration**

```
mqboolean FSMSG_readChar(StreamMessage msg, mqchar *value);
```

**Include**

```
stream_message.h
```

**Parameters**

Msg

StreamMessage

Value

Pointer to the mqchar, into which the unicode character are read

### **Return Values**

Mqboolean

TRUE if successful, FALSE if an error occurs

## **FSMSG\_readDouble**

### **Purpose**

Reads an mqdouble from the stream message.

### **Declaration**

```
mqboolean FSMSG_readDouble(StreamMessage msg, mqdouble *value);
```

### **Include**

stream\_message.h

### **Parameters**

Msg

StreamMessage

Value

Pointer to the mqdouble, into which data is read

### **Return Values**

Mqboolean

TRUE if successful, FALSE if an error occurs

## **FSMSG\_readFloat**

### **Purpose**

Reads an mqfloat from the stream message.

### **Declaration**

```
mqboolean FSMSG_readFloat(StreamMessage msg, mqfloat *value);
```

### **Include**

stream\_message.h



### Parameters

Msg

StreamMessage

Value

Pointer to the mqfloat, into which data is read

### Return Values

Mqboolean

TRUE if successful, FALSE if an error occurs

## FSMSG\_readInt

### Purpose

Reads an mqint from the stream message.

### Declaration

```
mqboolean FSMSG_readInt(StreamMessage msg, mqint *value);
```

### Include

stream\_message.h

### Parameters

Msg

StreamMessage

Value

Pointer to the mqint, into which data is read

### Return Values

Mqboolean

TRUE if successful, FALSE if an error occurs

## FSMSG\_readLong

### Purpose

Reads an mqlong from the stream message.

### **Declaration**

```
mqboolean FSMSG_readLong(StreamMessage msg, mqlong *value);
```

### **Include**

```
stream_message.h
```

### **Parameters**

Msg

StreamMessage

value

Pointer to the mqlong, into which data is read

### **Return Values**

Mqboolean

TRUE if successful, FALSE if an error occurs

## **FSMSG\_readShort**

### **Purpose**

Reads an mqshort from the stream message.

### **Declaration**

```
mqboolean FSMSG_readShort(StreamMessage msg, mqshort *value);
```

### **Include**

```
stream_message.h
```

### **Parameters**

Msg

StreamMessage

Value

Pointer to the mqshort, into which data is read

### **Return Values**

Mqboolean

TRUE if successful, FALSE if an error occurs

## FSMSG\_readString

### Purpose

Reads an mqstring from the stream message.

### Declaration

```
mqstring FSMSG_readString(StreamMessage msg);
```

### Include

```
stream_message.h
```

### Parameters

Msg

StreamMessage

### Return Values

Mqstring

mqstring from the stream message, NULL if an error occurs

## FSMSG\_reset

### Purpose

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

### Declaration

```
mqboolean FSMSG_reset(StreamMessage msg);
```

### Include

```
stream_message.h
```

### Parameters

Msg

StreamMessage

### Return Values

Mqboolean

TRUE if successful, FALSE if an error occurs

## FSMSG\_writeBoolean

### Purpose

Writes an mqboolean to the stream message, as a 1-byte value.

### Declaration

```
mqboolean FSMSG_writeBoolean(StreamMessage msg, mqboolean value);
```

### Include

```
stream_message.h
```

### Parameters

Msg

StreamMessage

Value

mqboolean to be written

### Return Values

Mqboolean

TRUE if successful, FALSE if an error occurs

## FSMSG\_writeByte

### Purpose

Writes an mqbyte to the stream message.

### Declaration

```
mqboolean FSMSG_writeByte(StreamMessage msg, mqbyte value);
```

### Include

```
stream_message.h
```

### Parameters

Msg

StreamMessage

Value

mqbyte to be written

**Return Values**

Mqboolean

TRUE if successful, FALSE if an error occurs

**FSMSG\_writeBytes****Purpose**

Writes an mqbyteArray 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**

```
mqboolean FSMSG_writeBytes(StreamMessage msg, mqbyteArray value, mqint length);
```

**Include**

stream\_message.h

**Parameters**

Msg

StreamMessage

Value

mqbyteArray to be written

length

Size of the mqbyteArray

**Return Values**

Mqboolean

TRUE if successful, FALSE if an error occurs

## FSMSG\_writeBytesWithOffset

### Purpose

Writes a portion of an mqbyteArray to the stream message. The portion of 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

```
mqboolean FSMSG_writeBytesWithOffset(StreamMessage msg, mqbyteArray value, mqint offset, mqint length);
```

### Include

```
stream_message.h
```

### Parameter

msg

StreamMessage

Value

mqbyteArray to be written

offset

Initial offset in the mqbyteArray

Length

Size of the mqbyteArray

### Return Values

Mqboolean

TRUE if successful, FALSE if an error occurs

## FSMSG\_writeChar

### Purpose

Writes an mqchar to the stream message.

### Declaration

```
mqboolean FSMSG_writeChar(StreamMessage msg, mqchar value);
```

### Include

```
stream_message.h
```

### Parameters

Msg

StreamMessage

Value

mqchar to be written

### Return Values

Mqboolean

TRUE if successful, FALSE if an error occurs

## FSMSG\_writeDouble

### Purpose

Writes an mqdouble to the stream message.

### Declaration

```
mqboolean FSMSG_writeDouble(StreamMessage msg, mqdouble value);
```

### Include

stream\_message.h

### Parameters

Msg

StreamMessage

Value

mqdouble to be written

### Return Values

Mqboolean

TRUE if successful, FALSE if an error occurs

## FSMSG\_writeFloat

### Purpose

Writes an mqfloat to the stream message.

### Declaration

```
mqboolean FSMSG_writeFloat(StreamMessage msg, mqfloat value);
```

### Include

stream\_message.h

### Parameters

Msg

StreamMessage

value

mqfloat to be written

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## FSMSG\_writeInt

### Purpose

Writes an mqint to the stream message.

### Declaration

```
mqboolean FSMSG_writeInt(StreamMessage msg, mqint value);
```

### Include

stream\_message.h

### Parameters

Msg

StreamMessage

Value

mqint to be written



### **Return Values**

Mqboolean

TRUE if successful, FALSE if an error occurs

## **FSMSG\_writeLong**

### **Purpose**

Writes an mqlong to the stream message.

### **Declaration**

```
mqboolean FSMSG_writeLong(StreamMessage msg, mqlong value);
```

### **Include**

stream\_message.h

### **Parameters**

Msg

StreamMessage

value

mqlong to be written

### **Return Values**

Mqboolean

TRUE if successful, FALSE if an error occurs

## **FSMSG\_writeShort**

### **Purpose**

Writes an mqshort to the stream message.

### **Declaration**

```
mqboolean FSMSG_writeShort(StreamMessage msg, mqshort value);
```

### **Include**

stream\_message.h

### **Parameters**

Msg

StreamMessage

Value

mqshort to be written

### **Return Values**

Mqboolean

TRUE if successful, FALSE if an error occurs

## **FSMSG\_writeString**

### **Purpose**

Writes an mqstring to the stream message.

### **Declaration**

```
mqboolean FSMSG_writeString(StreamMessage msg, mqstring value);
```

### **Include**

stream\_message.h

### **Parameters**

Msg

StreamMessage

Value

mqstring to be written

### **Return Values**

Mqboolean

TRUE if successful, FALSE if an error occurs

## **FTMSG\_getText**

### **Purpose**

Gets the string containing data of this message.

### **Declaration**

```
mqstring FTMSG_getText(TextMessage msg);
```

### **Include**

text\_message.h

### **Parameters**

Msg

TextMessage

### **Return Values**

Mqstring

mqstring containing data of this message

## **FTMSG\_setText**

### **Purpose**

Sets the string containing data of this message.

### **Declaration**

```
mqboolean FTMSG_setText(TextMessage msg, mqstring string);
```

### **Include**

text\_message.h

### **Parameters**

Msg

TextMessage

String

The mqstring containing data of this message

### **Return Values**

Mqboolean

TRUE if successful, FALSE if an error occurs

## HT\_Clear

### Purpose

Clears this hashtable so that it contains no keys and/or elements. This API would clear a hashtable, which only has values of type mqstring.

### Declaration

```
void HT_Clear(Hashtable table, mqboolean keyFlag);
```

### Include

```
fiorano_hashtable.h
```

### Parameters

Table

The hashtable that has to be cleared

keyFlag

True if both keys and values have to be cleared, false if only the values have to be cleared.

## HT\_ContainsKey

### Purpose

Tests if the specified mqstring is a key in this hashtable.

### Declaration

```
mqboolean HT_ContainsKey(Hashtable table, mqstring key);
```

### Include

```
fiorano_hashtable.h
```

### Parameters

Table

The hashtable for which the key has to be checked

Key

The possible hashtable key as mqstring

### Return Values

Mqboolean

True if and only if the specified object is a key in this hashtable, false otherwise.

## HT\_ContainsValue

### Purpose

Returns true if this hashtable maps one or more keys to this value.

### Declaration

```
mqboolean HT_ContainsValue(Hashtable table, mqobject value);
```

### Include

```
fiorano_hashtable.h
```

### Parameters

Table

The hashtable for which the value has to be checked

Value

The hashtable value as mqobject

### Return Values

Mqboolean

True if this hashtable maps one or more keys to the specified value.

## HT\_Get

### Purpose

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

### Declaration

```
mqobject HT_Get(Hashtable table, mqstring key);
```

### Include

```
fiorano_hashtable.h
```

### Parameters

Table

The hashtable from which the value has to be retrieved

Key

The hashtable key as mqstring

**Return Values**

Mqobject

The preceding value of the specified key in this hashtable, or NULL if it does not hold one.

## HT\_IsEmpty

**Purpose**

Tests if this hashtable maps no keys to values.

**Declaration**

```
mqobject HT_IsEmpty(Hashtable table);
```

**Include**

```
fiorano_hashtable.h
```

```
table
```

The hashtable to be checked

**Return Values**

Mqboolean

True if this hashtable maps no keys to values; false otherwise

## HT\_Put

**Purpose**

Maps the specified key to the specified value in the argument hashtable. Neither the key nor the value can be NULL. The value can be retrieved by calling the get method with a key that is equal to the original key.

**Declaration**

```
mqobject HT_Put(Hashtable table, mqstring key, mqobject value);
```

**Include**

```
fiorano_hashtable.h
```

**Parameters**

```
Table
```

The hashtable to which a name-value pair has to be added

### Key

The hashtable key

### Value

Value to be stored in the hashtable for the specified key

### Return Values

Mqobject

The preceding value of the specified key in this hashtable, or NULL if it does not hold one.

## HT\_RemoveElement

### Purpose

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

### Declaration

```
mqobject HT_RemoveElement(Hashtable table, mqstring key);
```

### Include

```
fiorano_hashtable.h
```

### Parameters

Table

The hashtable from which the value has to be removed

Key

The hashtable key that has to be removed

### Return Values

Mqobject

The value to which the key had been mapped in this hashtable, null if the key does not hold a mapping.

## HT\_Size

### Purpose

Returns the number of keys in this hashtable.

### Declaration

```
mqint HT_Size(Hashtable table);
```

### Include

```
fiorano_hashtable.h
```

### Parameters

Table

The hashtable whose size has to be returned

## IC\_Free

### Purpose

Frees up the resources allocated to the initial context structure.

### Declaration

```
void IC_free(InitialContext ic);
```

### Include

```
initial_context.h
```

### Parameters

Ic

The InitialContext that has to be cleaned up

## IC\_Lookup

### Purpose

Lookup an administered object from FioranoMQ server with name of the object passed as argument.

### Declaration

```
mqobject IC_Lookup(InitialContext ic, mqstring adminObjectName);
```



### **Include**

initial\_context.h

### **Parameters**

Ic

The InitialContext to used for lookup

adminObjectName

Name of the admin object to lookup

### **Return Values**

Mqobject

The admin object, in the form of mqobject

## **IC\_LookupQCF**

### **Purpose**

Lookup an Queue Connection Factory object from FioranoMQ server with name of the object passed as argument. This is used for server less mode. If server is present, it looks up as normal Lookup function.

### **Declaration**

```
mqobject IC_LookupQCF(InitialContext ic, mqstring adminObjectName);
```

### **Include**

initial\_context.h

### **Parameters**

ic

The InitialContext to used for lookup

adminObjectName

Name of the admin object to lookup

### **Return Values**

Mqobject

The admin object, in the form of mqobject

## IC\_LookupTCF

### Purpose

Lookup an administered object from FioranoMQ server with name of the object passed as argument. This is used for server less mode. If server is present, it looks up as normal Lookup function.

### Declaration

```
mqobject IC_LookupTCF(InitialContext ic, mqstring adminObjectName);
```

### Include

```
initial_context.h
```

### Parameters

ic

The InitialContext to used for lookup

adminObjectName

Name of the admin object to lookup

### Return Values

Mqobject

The admin object, in the form of mqobject

## MqExceptionClear

### Purpose

Clears information about the last occurred exception.

### Declaration

```
void MqExceptionClear ();
```

### Include

```
mq_errno.h
```

## MqExceptionOccurred

### Purpose

Checks if any exception or error occurred in any operations performed before the execution of this API.

### Declaration

```
mqboolean MqExceptionOccurred ();
```

### Include

```
mq_errno.h
```

### Return Values

True if an exception or error occurred; false otherwise

## MqGetLastErrCode

### Purpose

Returns the error code of the last occurred exception.

### Declaration

```
mqint MqGetLastErrCode ();
```

### Include

```
mq_errno.h
```

### Return Values

Error code of the last occurred error, as mqint

## MqPrintException

### Purpose

Prints the exception stack to stdout.

### Declaration

```
void MqPrintException();
```

### Include

```
mq_errno.h
```

## MqPrintExceptionToBuffer

### Purpose

Fills the exception stack trace into a pre-allocated buffer.

### Declaration

```
int MqPrintExceptionToBuffer(char* buffer, int size);
```

### Include

```
mq_errno.h
```

### Parameters

char\*

Pre-allocated buffer into which the exception stack has to be read

int

Size of the pre-allocated buffer

### Return

Returns the size of the exception stack trace

## MSG\_acknowledge

### Purpose

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 consumed messages of its JMS session 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 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

```
mqboolean MSG_acknowledge(Message msg);
```

### **Include**

message.h

### **Parameters**

msg

Message

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **MSG\_clearBody**

### **Purpose**

Clears out the message body. Clearing the body of a message does not clear its header values or property entries.

### **Declaration**

```
mqboolean MSG_clearBody(Message msg);
```

### **Include**

message.h

### **Parameters**

msg

Message

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **MSG\_clearProperties**

### **Purpose**

Clears the properties of the message. The message header fields and body are not cleared.

### **Declaration**

```
mqboolean MSG_clearProperties(Message msg);
```

**Include**

message.h

**Parameters**

msg

Message

**Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## MSG\_free

**Purpose**

Frees the resources allocated for the message. This API is to be used to free all message types.

**Declaration**

```
void MSG_free(Message msg);
```

**Include**

message.h

**Parameters**

msg

Message

## MSG\_getBooleanProperty

**Purpose**

Gets the mqboolean property value with the given name.

**Declaration**

```
mqboolean MSG_getBooleanProperty(Message msg, mqstring propName,  
mqboolean *value);
```

**Include**

message.h

### Parameters

msg

Message

propName

Name of the mqboolean property

value

Pointer to mqboolean, into which the mqboolean property is to be read

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## MSG\_getByteProperty

### Purpose

Gets the mqbyte property value with the given name.

### Declaration

```
mqboolean MSG_getByteProperty(Message msg, mqstring propName,  
mqbyte *value);
```

### Include

message.h

### Parameters

msg

Message

propName

Name of the mqbyte property

value

Pointer to mqbyte, into which the mqbyte property is to be read

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## MSG\_getDoubleProperty

### Purpose

Gets the mqdouble property value with the given name.

### Declaration

```
mqboolean MSG_getDoubleProperty(Message msg, mqstring propName,  
mqdouble *value);
```

### Include

message.h

### Parameters

msg

Message

propName

Name of the mqdouble property

Value

Pointer to mqdouble, into which the mqdouble property is to be read

### Return Values

Mqboolean

TRUE if successful, FALSE if an error occurs

## MSG\_getFloatProperty

### Purpose

Gets the mqfloat property value with the given name.

### Declaration

```
mqboolean MSG_getFloatProperty(Message msg, mqstring propName,  
mqfloat *value);
```

### Include

message.h

### Parameters

msg



Message

propName

Name of the mqfloat property

value

Pointer to mqfloat, into which the mqfloat property is to be read

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **MSG\_getIntProperty**

### **Purpose**

Gets the mqint property value with the given name.

### **Declaration**

```
mqboolean MSG_getIntProperty(Message msg, mqstring propName, mqint *value);  
message.h
```

### **Parameters**

msg

Message

propName

Name of the mqint property

value

Pointer to mqint, into which the mqint property is to be read

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## MSG\_getJMSCorrelationID

### Purpose

Gets the correlation ID of the message as mqstring.

### Declaration

```
mqstring MSG_getJMSCorrelationID(Message msg);
```

### Include

message.h

### Parameters

msg

Message

### Return Values

mqstring

Correlation ID of the message, NULL if an error occurs

## MSG\_getJMSCorrelationIDAsBytes

### Purpose

Gets the correlation ID of the message as mqbyteArray.

### Declaration

```
mqint MSG_getJMSCorrelationIDAsBytes(Message msg, mqbyteArray *cor-relationID);
```

### Include

message.h

### Parameters

msg

Message

correlationID

mqbyteArray where the correlationID has to be set

### Return Values

mqint

Length of the mqbyteArray, -1 if an error occurs

## MSG\_getJMSDeliveryMode

### Purpose

Gets the delivery mode for this message.

### Declaration

```
mqboolean MSG_getJMSDeliveryMode(Message msg, mqint *value);
```

### Include

message.h

### Parameters

msg

Message

value

Pointer to the mqint, into which the delivery mode is to be read

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## MSG\_getJMSDestination

### Purpose

Gets the destination for this message.

### Declaration

```
struct _Destination* MSG_getJMSDestination(Message msg);
```

### Include

message.h

### Parameters

msg

Message

### **Return Values**

struct \_Destination\*

Destination of the message

## **MSG\_getJMSExpiration**

### **Purpose**

Gets the expiration time of the message.

### **Declaration**

```
mqboolean MSG_getJMSExpiration(Message msg, mqlong *value);
```

### **Include**

message.h

### **Parameters**

msg

Message

value

Pointer to mqlong, into which the message expiration time is to be read

### **Return Values**

mqboolean

## **MSG\_getJMSMessageID**

### **Purpose**

Gets the message ID for this message.

### **Declaration**

```
mqstring MSG_getJMSMessageID(Message msg);
```

### **Include**

message.h

### **Parameters**

msg

Message

## Return Values

mqstring

Message ID as mqstring

## MSG\_getJMSPriority

### Purpose

Gets the message priority.

### Declaration

```
mqboolean MSG_getJMSPriority(Message msg, mqint *value);
```

### Include

message.h

### Parameters

msg

Message

value

Pointer to mqint, into which the message priority is to be read

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## MSG\_getJMSRedelivered

### Purpose

Gets an indication of redelivery of this message.

### Declaration

```
mqboolean MSG_getJMSRedelivered(Message msg, mqboolean *value);
```

### Include

message.h

### Parameters

msg

Message

value

Pointer to mqboolean, into which the redelivered flag is set

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **MSG\_getJMSReplyTo**

### **Purpose**

Gets the destination structure to which a reply to this message should be sent.

### **Declaration**

```
struct _Destination* MSG_getJMSReplyTo(Message msg);
```

### **Include**

message.h

### **Parameters**

msg

### **Return Values**

struct \_Destination\*

Destination object, where a response to this message is to be sent

## **MSG\_getJMSTimestamp**

### **Purpose**

Gets the message timestamp for this message.

### **Declaration**

```
mqboolean MSG_getJMSTimestamp(Message msg, mqlong *value);
```

### **Include**

message.h

### **Parameters**

msg

Message

value

Pointer to the mqlong, into which timestamp is to be read

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **MSG\_getJMSType**

### **Purpose**

Gets the message type.

### **Declaration**

```
mqstring MSG_getJMSType(Message msg);
```

### **Include**

message.h

### **Parameters**

msg

Message

### **Return Values**

mqstring

Message type as an mqstring

## **MSG\_getLongProperty**

### **Purpose**

Gets the mqlong property value with the given name.

### **Declaration**

```
mqboolean MSG_getLongProperty(Message msg, mqstring propName, mqlong *value);
```

### **Include**

message.h

### Parameters

msg

Message

propName

Name of the mqlong property

value

Pointer to mqlong, into which the mqlong property is to be read

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## MSG\_getObjectProperty

### Purpose

Gets an Object Property from the message.

### Declaration

```
mqobject MSG_getObjectProperty(Message msg, mqstring propName);
```

### Include

message.h

### Parameters

msg

Message from which the property is to be read

propName

Name of the property

### Return Values

mqobject

The mqobject property



## MSG\_getPropertyNames

### Purpose

Gets an array of all the property names.

### Declaration

```
mqint MSG_getPropertyNames(Message msg, mqstring** list);
```

### Include

message.h

### Parameters

msg

Message

list

Pointer to an array of strings, into which the property names are set

### Return Values

mqint

Number of properties, -1 if an error occurs

## MSG\_getShortProperty

### Purpose

Gets the mqshort property value with the given name.

### Declaration

```
mqboolean MSG_getShortProperty(Message msg, mqstring propName, mqshort *value);
```

### Include

message.h

### Parameters

msg

Message

propName

Name of the mqshort property

value

Pointer to mqshort, into which the mqshort property is to be read

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **MSG\_getStringProperty**

### **Purpose**

Gets the mqstring property value with the given name.

### **Declaration**

```
mqstring MSG_getStringProperty(Message msg, mqstring propName);
```

### **Include**

message.h

### **Parameters**

msg

Message Name of the mqstring property

### **Return Values**

mqstring

mqstring property with the given name, NULL if it does not exist

## **MSG\_propertyExists**

### **Purpose**

Checks if a property value exists.

### **Declaration**

```
mqboolean MSG_propertyExists(Message msg, mqstring propName, mqboolean *value);
```

### **Include**

message.h

### **Parameters**

msg

Message

propName

Name of the property to test

value

Pointer to mqboolean, which is used to report whether or not the property exists

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **MSG\_setBooleanProperty**

### **Purpose**

Sets an mqboolean property value, into the Message, with the given name.

### **Declaration**

```
mqboolean MSG_setBooleanProperty(Message msg, mqstring propName, mqboolean propValue);
```

### **Include**

message.h

### **Parameters**

msg

Message

propName

Name of the mqboolean property

propValue

mqboolean property value to set in the message

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## MSG\_setByteProperty

### Purpose

Sets an mqbyte property value, into the Message, with the given name.

### Declaration

```
mqboolean MSG_setByteProperty(Message msg, mqstring propName, mqbyte propValue);
```

### Include

message.h

### Parameters

msg

Message

propName

Name of the mqbyte property

propValue

mqbyte property value to set in the message

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## MSG\_setDoubleProperty

### Purpose

Sets an mqdouble property value, into the Message, with the given name.

### Declaration

```
mqboolean MSG_setDoubleProperty(Message msg, mqstring propName, mqdouble propValue);
```

### Include

message.h

### Parameters

msg

Message

propName

Name of the mqdouble property

propValue

mqdouble property value to set in the message

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **MSG\_setFloatProperty**

### **Purpose**

Sets an mqfloat property value, into the Message, with the given name.

### **Declaration**

```
mqboolean MSG_setFloatProperty(Message msg, mqstring propName, mqfloat propValue);
```

message.h

### **Parameters**

msg

Message

propName

Name of the mqfloat property

propValue

mqfloat property value to set in the message

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **MSG\_setIntProperty**

### **Purpose**

Sets an mqint property value, into the Message, with the given name.

### **Declaration**

```
mqboolean MSG_setIntProperty(Message msg, mqstring propName, mqint propValue);
```

### **Include**

message.h

### **Parameters**

msg

Message

propName

Name of the mqint property

propValue

mqint property value to set in the message

### **Return Values**

mqboolean

## **MSG\_setJMSCorrelationID**

### **Purpose**

Sets the correlation ID for this message.

### **Declaration**

```
mqboolean MSG_setJMSCorrelationID(Message msg, mqstring correlationID);
```

### **Include**

message.h

### **Parameters**

msg

Message

correlationID

Correlation ID to be set for the message, specified as mqstring

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## MSG\_setJMSCorrelationIDAsBytes

### Purpose

Sets the correlation ID for this message as mqbyteArray.

### Declaration

```
mqboolean MSG_setJMSCorrelationIDAsBytes(Message msg, mqbyteArray correlationID, mqint length);
```

### Include

message.h

### Parameters

msg

Message Correlation ID to be set for the message, specified as mqbytearray

length

Size of the mqbyteArray correlationID

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## MSG\_setJMSDeliveryMode

### Purpose

Sets the delivery mode for this message.

### Declaration

```
mqboolean MSG_setJMSDeliveryMode(Message msg, mqint mode);
```

### Include

message.h

### Parameters

msg

Message

mode

The delivery mode for this message

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **MSG\_setJMSDestination**

### **Purpose**

Sets the destination for this message.

### **Declaration**

```
mqboolean MSG_setJMSDestination(Message msg, struct _Destination*dest);
```

### **Includes**

message.h

### **Parameters**

msg

Message

dest

Destination for this message

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **MSG\_setJMSExpiration**

### **Purpose**

Sets the expiration value of the message. This API is used only by FioranoMQ C-rtl to set the expiration time of the message.

### **Declaration**

```
mqboolean MSG_setJMSExpiration(Message msg, mqlong expiration);
```

### **Include**

message.h



### Parameters

msg

Message

expiration

Expiration time of the message

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## MSG\_setJMSMessageID

### Purpose

Sets the message ID for this message.

### Declaration

```
mqboolean MSG_setJMSMessageID(Message msg, mqstring str);
```

### Include

message.h

### Parameters

msg

Message

str

ID of the message

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## MSG\_setJMSPriority

### Purpose

Sets the priority for this message.

### Declaration

```
mqboolean MSG_setJMSPriority(Message msg, mqint priority);
```

### Include

message.h

### Parameters

msg

Message

priority

The priority of this message specified as mqint

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## MSG\_setJMSRedelivered

### Purpose

Set to indicate whether this message is being redelivered.

### Declaration

```
mqboolean MSG_setJMSRedelivered(Message msg, mqboolean redelivered);
```

### Include

message.h

### Parameters

msg

Message

redelivered

TRUE if the message is being redelivered, FALSE otherwise

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **MSG\_setJMSReplyTo**

### **Purpose**

Sets the destination, where a reply to this message should be sent.

### **Declaration**

```
mqboolean MSG_setJMSReplyTo(Message msg, struct _Destination*replyTo);
```

### **Include**

message.h

### **Parameters**

msg

Message

struct \_Destination\*

replyTo destination structure to which the response should be sent

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **MSG\_setJMSTimestamp**

### **Purpose**

Sets the message timestamp as mqlong.

### **Declaration**

```
mqboolean MSG_setJMSTimestamp(Message msg, mqlong timeStamp);
```

### **Include**

message.h

### **Parameters**

msg

Message

timeStamp

Timestamp for this message, specified as mqlong

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **MSG\_setJMSType**

### **Purpose**

Sets the message type.

### **Declaration**

```
mqboolean MSG_setJMSType(Message msg, mqstring type);
```

### **Include**

message.h

### **Parameters**

msg

Message

type

The type of message

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **MSG\_setLongProperty**

### **Purpose**

Sets an mqlong property value, into the Message, with the given name.

### **Declaration**

```
mqboolean MSG_setLongProperty(Message msg, mqstring propName, mqlong propValue);
```

### **Include**

message.h

### **Parameters**

msg

Message

propName

Name of the mqlong property

propValue

mqlong property value to set in the message

### **Return Values**

mqboolean

## **MSG\_setObjectProperty**

### **Purpose**

Sets an Object as a property on a message.

### **Declaration**

```
mqboolean MSG_setObjectProperty(Message msg, mqstring propName, mqobject propValue,  
mqint size);
```

### **Include**

message.h

### **Parameters**

msg

The message on which the property is to be set

propName

Name of the Property to be set

### **Parameters**

propValue

The Object that is to be set as property

size

Size of the Object taken as an integer

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **MSG\_setShortProperty**

### **Purpose**

Sets an mqshort property value, into the Message, with the given name.

### **Declaration**

```
mqboolean MSG_setShortProperty(Message msg, mqstring propName, mqshort propValue);  
message.h
```

### **Parameters**

msg

Message

propName

Name of the mqshort property

propValue

mqshort property value to set in the message

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **MSG\_setStringProperty**

### **Purpose**

Sets an mqstring property value, into the Message, with the given name.

### **Declaration**

```
mqboolean MSG_setStringProperty(Message msg, mqstring propName, mqstring propValue);
```

### **Include**

message.h

**Parameters**

msg

Message

propName

Name of the mqstring property

propValue

mqstring property value to set in the message

**Return Values**

mqboolean

## newHashtable

**Purpose**

Constructs a new, empty hashtable with a default initial capacity (11) and load factor, which is 0.75.

**Declaration**

```
Hashtable newHashtable();
```

**Include**

```
fiorano_hashtable.h
```

**Return Values**

Hashtable

The empty hashtable structure

## newInitialContext\_env

**Purpose**

Constructs the Initial Context for lookup of Admin Objects using the hashtable passed as argument. This hashtable has all the required environment properties passed as name-value pairs.

**Declaration**

```
InitialContext newInitialContext_env(struct _Hashtable* env);
```

### **Include**

initial\_context.h

### **Parameters**

env

The Hashtable structure containing environment parameters such as url, user-name, password

### **Return Values**

InitialContext

A newly created InitialContext structure, NULL if an error occurs

## **newInitialContext\_HTTP**

### **Purpose**

Constructs the Initial Context for lookup of Admin Objects, using the HTTP protocol for communication.

### **Declaration**

```
InitialContext newInitialContext_HTTP(mqstring url, mqstring username, mqstring passwd);
```

### **Include**

initial\_context.h

### **Parameters**

url

URL of the FioranoMQ Server

username

The user name to be used for lookup

passwd

The password for the username

### **Return Values**

InitialContext

A newly created InitialContext structure, NULL if an error occurs



## newInitialContext\_SSL

### Purpose

Constructs the Initial Context for lookup of Admin Objects using the SSL protocol. The various SSL parameters are passed as arguments to this API.

### Declaration

```
InitialContext newInitialContext_SSL(mqstring url, mqstring user-name, mqstring  
passwd, mqstring certFile, mqstring privateKeyFile, mqstring keyPass);
```

### Include

```
initial_context.h  
url
```

URL of the FioranoMQ Server

```
username
```

The user name to be used for lookup

```
passwd
```

The password for the user

```
certFile
```

Fully qualified path of certificate file

```
privateKeyFile
```

Fully qualified path of private key file

```
keyPass
```

Password under which private key file is encrypted

### Return Values

```
InitialContext
```

A newly created InitialContext structure, NULL if an error occurs

## newInitialContext1

### Purpose

Constructs the Initial Context for lookup of Admin Objects using the serverName, port, username and password passed as arguments.

### Declaration

```
InitialContext newInitialContext1(mqstring serverName, mqint port, mqstring username, mqstring passwd);
```

### Include

```
initial_context.h
```

### Parameters

address

Name or IP Address of the machine on which FioranoMQ is running

port

Port number on which FioranoMQ Server is running

Username

The user name to be used for lookup

passwd

Password for the username

### Return Values

InitialContext

A newly created InitialContext structure, NULL if an error occurs

## newInitialContext

### Purpose

Constructs the Initial Context for lookup of Admin Objects using the url, username and password that are passed as parameters.

### Declaration

```
InitialContext newInitialContext(mqstring url, mqstring username, mqstring passwd);
```

### Include

```
initial_context.h
```

### Parameters

url

URL of FioranoMQ Server that includes the address and port

username

The user name that is used for lookup

passwd

Password for the username

### Return Values

InitialContext

A newly created InitialContext structure, NULL if an error occurs

## newInitialContextDefaultParams

### Purpose

Creates the Initial Context for lookup of Admin Objects using the the default values for address, port number, username and password. The default values are as follows:

address = localhost

port = 1856

username = anonymous

password = anonymous

### Declaration

```
InitialContext newInitialContextDefaultParams();
```

### Include

initial\_context.h

### Return Values

InitialContext

A newly created InitialContext structure, NULL if an error occurs

## newQueueRequestor

### Purpose

Creates a new QueueRequestor on the specified Queue. This implementation assumes the session parameter to be non-transacted, with a delivery mode of AUTO\_ACKNOWLEDGE.

### Declaration

```
QueueRequestor newQueueRequestor(QueueSession session, Queue queue);
```

### Include

```
queue_requestor.h
```

### Parameters

session

The queue session

queue

The queue on which the request/reply call is to be performed

### Return Values

QueueRequestor

Newly created QueueRequestor

## newTopicRequestor

### Purpose

Creates a TopicRequestor.

This implementation assumes the session parameter to be non-transacted, with a delivery mode of AUTO\_ACKNOWLEDGE.

### Declaration

```
TopicRequestor newTopicRequestor(TopicSession session, Topic topic);
```

### Include

```
topic_requestor.h
```

### Parameters

session

The topic session to which the topic belongs

topic

The topic on which the request/reply call is to be performed

### **Return Values**

TopicRequestor

Newly created TopicRequestor, NULL if an error occurs

## **Q\_free**

### **Purpose**

Frees all the resources allocated on behalf of a queue structure.

### **Declaration**

```
void Q_free(Queue queue);
```

### **Include**

queue.h

### **Parameters**

queue

The Queue structure that has to be freed

## **Q\_getQueueName**

### **Purpose**

Gets name of the queue passed as argument.

### **Declaration**

```
mqstring Q_getQueueName(Queue queue);
```

### **Include**

queue.h

### **Parameters**

queue

Queue whose name is desired

### **Return Values**

mqstring

The queue name

## QBE\_free

### Purpose

Frees all the resources allocated on behalf of QueueBrowserEnumeration.

### Declaration

```
void QBE_free(QueueBrowserEnumeration browserEnum);
```

### Include

```
queue_browser_enumeration.h
```

### Parameters

browserEnum

QueueBrowserEnumeration that has to be cleaned up

## QBE\_hasMoreElements

### Purpose

Checks if the queue browser enumeration contains more elements that is, messages.

### Declaration

```
mqboolean QBE_hasMoreElements(QueueBrowserEnumeration browserE-num);
```

### Include

```
queue_browser_enumeration.h
```

### Parameters

browserEnum

QueueBrowserEnumeration that has to be checked for more elements

### Return Values

mqboolean

TRUE if and only if QueueBrowserEnumeration contains atleast one element; false otherwise

## QBE\_nextElement

### Purpose

Gets the next message available with the queue browser enumeration.

### Declaration

```
Message QBE_nextElement(QueueBrowserEnumeration browserEnum);
```

### **Include**

queue\_browser\_enumeration.h

### **Parameters**

browserEnum

QueueBrowserEnumeration whose next message is desired

### **Return Values**

Message

The next available message in the enumeration; NULL if an error occurs

## **QBWSR\_close**

### **Purpose**

Closes all the resources on behalf of a QueueBrowser.

### **Declaration**

```
mqboolean QBWSR_close(QueueBrowser browser);
```

### **Include**

queue\_browser.h

### **Parameters**

browser

The QueueBrowser that has to be closed

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **QBWSR\_free**

### **Purpose**

Frees the resources allocated for the QueueBrowser.

### **Declaration**

```
void QBWSR_free(QueueBrowser browser);
```

### **Include**

queue\_browser.h

## Parameters

browser

QueueBrowser that has to be cleaned up

## QBWSR\_getEnumeration

### Purpose

Gets an enumeration for browsing the current queue messages in the same order as they would be received.

### Declaration

```
struct _QueueBrowserEnumeration* QBWSR_getEnumeration(QueueBrowser browser);
```

### Include

queue\_browser.h

### Parameters

browser

QueueBrowser using which the Enumeration is being obtained

### Return Values

```
struct _QueueBrowserEnumeration*
```

QueueBrowserEnumeration for browsing the messages

## QBWSR\_getMessageSelector

### Purpose

Gets message selector expression of this queue browser.

### Declaration

```
mqstring QBWSR_getMessageSelector(QueueBrowser browser);
```

### Include

queue\_browser.h

### Parameters

browser

QueueBrowser whose message selector is desired



## Return Values

mqstring

The message selector of queue browser

## QBWSR\_getQueue

### Purpose

Get the queue on which this queue browser is created.

### Declaration

```
Queue QBWSR_getQueue(QueueBrowser browser);
```

### Include

queue\_browser.h

### Parameters

browser

QueueBrowser whose Queue is desired

### Return Values

Queue

The Queue over which the QueueBrowser is created

## QC\_close

### Purpose

Closes the connection. 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

```
mqboolean FBMSG_readBoolean(BytesMessage msg, mqboolean* value);
```

### Include

queue\_connection.h

### Parameters

connection

QueueConnection that has to be closed

### Return Values

mqboolean

## QC\_createQueueSession

### Purpose

Creates a QueueSession with the given mode.

### Declaration

```
QueueSession QC_createQueueSession(QueueConnection connection, mqboolean transacted,  
mqint acknowledgeMode);
```

### Include

queue\_connection.h

### Parameters

connection

QueueConnection

mqboolean

Indicates if the session is transacted. If TRUE, the session is transacted.

acknowledgeMode

Indicates whether the consumer or the client acknowledges any messages it receives; ignored if the session is transacted. Legal values AUTO\_ACKNOWLEDGE, CLIENT\_ACKNOWLEDGE and DUPS\_OK\_ACKNOWLEDGE.

### Return Values

QueueSession

A newly created QueueSession, NULL if an error occurs

For values of the different acknowledgement modes, refer to the chapter "C-rtl Constants".

## QC\_free

### Purpose

Frees all the resources allocated for the QueueConnection.

### Declaration

```
void QC_free(QueueConnection connection);
```

### Include

```
queue_connection.h
```

### Parameters

connection

QueueConnection that has to be cleaned up

## QC\_getClientID

### Purpose

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

### Declaration

```
mqstring QC_getClientID(QueueConnection connection);
```

### Include

```
queue_connection.h
```

### Parameters

connection

The QueueConnection for which the clientID is desired

### Return Values

mqstring

The unique client identifier for this connection

## QC\_getExceptionListener

### Purpose

Returns the ExceptionListener structure for this Queue connection.

### Declaration

```
ExceptionListener QC_getExceptionListener(QueueConnection connection);  
queue_connection.h
```

### Parameters

connection

The QueueConnection for which the exception listener is desired

### Return Values

ExceptionListener

The ExceptionListener for this connection

For details, refer to the chapter "ExceptionListener".

## QC\_setClientID

### Purpose

Sets the client identifier for this connection. The preferred way to assign a client identifier of a JMS client is to configure it in a client-specific ConnectionFactory object, which is then transparently assigned to the Connection object it creates.

The purpose of the client identifier is to associate a connection and its objects with a state maintained on behalf of the client by a provider. The only such state identified by the JMS API is that required to support durable subscriptions.

If another connection with the same clientID is running, when this method is called, an error is returned.

### Declaration

```
mqboolean QC_setClientID(QueueConnection connection, mqstring clientID);
```

### Include

queue\_connection.h

### Parameters

connection

QueueConnection for which the clientID is to be set

clientID

The unique client identifier

**Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## QC\_setExceptionListener

**Purpose**

Sets an exception listener for this connection. If the FioranoMQ server detects a serious problem with a connection, it notifies the ExceptionListener of the connection, if one has been registered. This is done by calling the onException method of the listener.

**Declaration**

```
mqboolean QC_setExceptionListener(QueueConnection connection, ExceptionListenerPointer ptrListener, void* param);
```

**Include**

queue\_connection.h

**Parameters**

connection

QueueConnection for which the ExceptionListener is to be set

ptrListener

The function pointer of the callback function for the ExceptionListener to be registered

param

Any parameter that is required by the client application in the onException callback of the ExceptionListener

**Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

For details, refer to the chapter "ExceptionListener".

## QC\_start

### Purpose

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

### Declaration

```
mqboolean QC_start(QueueConnection connection);
```

### Include

```
queue_connection.h
```

### Parameters

```
connection
```

QueueConnection that has to be started

### Return Values

```
mqboolean
```

TRUE if successful, FALSE if an error occurs

## QC\_stop

### Purpose

Temporarily stops the delivery of incoming messages of a connection. Delivery can be restarted using the start method of a connection. When the connection is stopped, delivery to all the message consumers of the connection 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. Stopping a connection has no effect on its ability to send messages. A call to stop on a connection that has already been stopped is ignored.

### Declaration

```
mqboolean QC_stop(QueueConnection connection);
```

### Include

```
queue_connection.h
```

### Parameters

```
connection
```

QueueConnection that has to be stopped

**Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## QCF\_createQueueConnection

**Purpose**

Creates a queue connection with specified user identity. The connection is created in stopped mode. No messages are delivered until the QC\_start method is explicitly called.

**Declaration**

```
QueueConnection QCF_createQueueConnection(QueueConnectionFactory qcf, mqstring  
userName, mqstring password);
```

**Include**

queue\_connection\_factory.h

**Parameters**

qcf

QueueConnectionFactory using which the QueueConnection is to be created

userName

User name to be used for creating the QueueConnection

password

Password for the specified user

**Return Values**

QueueConnection

A newly created QueueConnection, NULL if an error occurs

## QCF\_createQueueConnectionDefParams

**Purpose**

Creates a queue connection with default user identity. The connection is created in stopped mode. No messages are delivered until the QC\_start method is explicitly called.

**Declaration**

```
QueueConnection QCF_createQueueConnectionDefParams(QueueConnectionFactory qcf);  
queue_connection_factory.h
```

**Parameters**

qcf

QueueConnectionFactory using which the QueueConnection is to be created

**Return Values**

QueueConnection

A newly created QueueConnection, NULL if an error occurs

## QCF\_free

**Purpose**

Frees all the resources allocated for the QueueConnectionFactory.

**Declaration**

```
void QCF_free(QueueConnectionFactory qcf);
```

**Include**

queue\_connection\_factory.h

**Parameters**

qcf

QueueConnectionFactory that has to be freed

## QRCVR\_close

**Purpose**

Closes the queue receiver and the resources allocated on behalf of the QueueReceiver.

This call blocks until it receives the message, or message listener in progress has completed. A blocked queue receiver call returns null when this queue receiver is closed.

**Declaration**

```
mqboolean QRCVR_close(QueueReceiver receiver);  
queue_connection_factory_metadata.h
```

**Parameters**

receiver

QueueReceiver that is to be closed

**Return Values**

mqboolean



TRUE if successful, FALSE if an error occurs

## QRCVR\_free

### Purpose

Frees the resources assigned to a queue receiver.

### Declaration

```
void QRCVR_free(QueueReceiver receiver);
```

### Include

```
queue_connection_factory_metadata.h
```

### Parameters

receiver

QueueReceiver that is to be freed

## QRCVR\_getMessageListener

### Purpose

Gets the MessageListener of the queue receiver

### Declaration

```
MessageListener QRCVR_getMessageListener(QueueReceiver receiver);
```

### Include

```
queue_connection_factory_metadata.h
```

### Parameters

receiver

### Return Values

MessageListener

Listener for the message consumer, NULL if no listener is set

Also, refer to the section "QRCVR\_setMessageListener"

## QRCVR\_getMessageSelector

### Purpose

Gets the message selector expression of this queue receiver.

### Declaration

```
mqstring QRCVR_getMessageSelector(QueueReceiver receiver);
```

### Include

```
queue_connection_factory_metadata.h
```

### Parameters

receiver

QueueReceiver whose message selector is desired

### Return Values

mqstring

Message selector of this queue receiver as mqstring, NULL if no message selector is set or message selector is set as NULL

## QRCVR\_getQueue

### Purpose

Gets the queue associated with this queue receiver.

### Declaration

```
Queue QRCVR_getQueue(QueueReceiver receiver);  
queue_connection_factory_metadata.h
```

### Parameters

receiver

QueueReceiver whose queue is desired

### Return Values

Queue

Queue of this receiver

## QRCVR\_receive

### Purpose

Receives the next message produced for this queue receiver. This call blocks indefinitely until a message is produced or this queue receiver is closed.

If this receive is done within a transaction, the receiver retains the message until the transaction commits.

### Declaration

```
Message QRCVR_receive(QueueReceiver receiver);
```

### Include

```
queue_connection_factory_metadata.h
```

### Parameters

receiver

QueueReceiver on which the receive has to be called

### Return Values

Message

The next message produced for this queue receiver, NULL if this queue receiver is concurrently closed.

## QRCVR\_receiveNoWait

### Purpose

Receives the next message if one is immediately available.

### Declaration

```
Message QRCVR_receiveNoWait(QueueReceiver receiver);
```

### Include

```
queue_receiver.h
```

### Parameters

receiver

QueueReceiver on which the receive call is made

### Return Values

Message

The next message produced for this message consumer, NULL if one is not available.

## QRCVR\_receiveWithTimeout

### Purpose

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

### Declaration

```
Message QRCVR_receiveWithTimeout(QueueReceiver receiver, mqlong timeout);
```

### Include

```
queue_connection_factory_metadata.h
```

### Parameters

receiver

QueueReceiver on which the receive call is made

timeout

The timeout value (in milliseconds)

### Return Values

Message

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

## QRCVR\_setFioranoMessageListener

### Purpose

Sets MessageListener of the message consumer. This API can be used to set a FioranoMessageListener callback pointer that can be passed a parameter, which in turn can be used in the implementation of the callback function.

### Declaration

```
mqboolean QRCVR_setFioranoMessageListener(QueueReceiver consumer,  
FioranoMessageListener ptrmessageListener, void* param);
```

### Include

```
queue_receiver.h
```

### Parameters

consumer

QueueReceiver whose message listener is to be set

ptrmessageListener

Listener to which the messages are to be delivered. This listener accepts a Message structure and a void\* parameter as its arguments.

param

Parameter to be used in the MessageListener callback

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## QRCVR\_setMessageListener

### Purpose

Sets MessageListener of the message consumer.

### Declaration

```
mqboolean QRCVR_setMessageListener(QueueReceiver receiver, MessageListener listener);
```

### Include

queue\_receiver.h

### Parameter

receiver

QueueReceiver whose message listener is to be set

messageListener

Listener to which the messages are to be delivered

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## QRQST\_close

### Purpose

Closes the queue requestor and all the resources allocated on behalf of the QueueRequestor.

### Declaration

```
void QRQST_close(QueueRequestor requestor);
```

### Include

queue\_requestor.h

### Parameters

requestor

QueueRequestor that has to be closed

## QRQST\_free

### Purpose

Frees the resources allocated on behalf of the QueueRequestor.

### Declaration

```
void QRQST_free(QueueRequestor requestor);
```

### Include

queue\_requestor.h

### Parameters

requestor

## QRQST\_request

### Purpose

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

### Declaration

```
Message QRQST_request(QueueRequestor requestor, Message message);
```

### Include

queue\_requestor.h

### Parameters

requestor

QueueRequestor on which the request has to be made

message

The message to be sent

### Return Values

Message

The reply message, NULL if an error occurs

## QRQST\_requestWithTimeout

### Purpose

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

### Declaration

```
Message QRQST_requestWithTimeout(QueueRequestor requestor, Message message, mqlong timeout);
```

### Include

```
queue_requestor.h
```

### Parameters

requestor

QueueRequestor on which the request is to be made

message

The message to be sent

timeout

The timeout period for which the requestor waits for a reply

### Return Values

Message

The reply message, NULL if an error occurs

## QS\_close

### Purpose

Closes the queue session and resources allocated on behalf of the QueueSession.

### Declaration

```
mqboolean QS_close(QueueSession session);
```

### Include

```
queue_session.h
```

### Parameters

session

QueueSession that has to be closed

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## QS\_commit

### Purpose

Commits all messages done in this transaction and releases any locks held at the time of call of this API.

### Declaration

```
mqboolean QS_commit(QueueSession session);
```

### Include

```
queue_session.h
```

### Parameters

session

QueueSession that has to be committed

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs



## QS\_createBrowser

### Purpose

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

### Declaration

```
QueueBrowser QS_createBrowser(QueueSession session, Queue queue);
```

### Include

queue\_session.h

### Parameters

session

QueueSession using which the browser is to be created

queue

The queue to be accessed

### Return Values

QueueBrowser

Newly created QueueBrowser, NULL if an error occurs

## QS\_createBrowserWithSelector

### Purpose

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

### Declaration

```
QueueBrowser QS_createBrowserWithSelector(QueueSession session, Queue queue, mqstring messageSelector);
```

### Include

queue\_session.h

### Parameters

session

QueueSession using which the browser is to be created

queue

The queue to be accessed

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 browser.

### **Return Values**

QueueBrowser

Newly created QueueBrowser, NULL if an error occurs

## **QS\_createBytesMessage**

### **Purpose**

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

### **Declaration**

```
Message QS_createBytesMessage(QueueSession session);
```

### **Include**

queue\_session.h

### **Parameters**

session

QueueSession using which the bytes message is to be created.

### **Return Values**

Message

A newly created BytesMessage, NULL if an error occurs

## **QS\_createMapMessage**

### **Purpose**

Creates a map message. A MapMessage object is used to send a self-defining set of name-value pairs, where names are of type mqstring and values can be different datatypes.

### **Declaration**

```
Message QS_createMapMessage(QueueSession session);
```

### **Include**

queue\_session.h

### **Parameters**

session

QueueSession using which the map message is to be created

### **Return Values**

Message

A newly created MapMessage, NULL if an error occurs

## **QS\_createQueue**

### **Purpose**

Creates a queue identity given a Queue name.

### **Declaration**

```
Queue QS_createQueue(QueueSession session, mqstring queueName);  
queue_session.h
```

### **Parameters**

session

QueueSession using which the queue is to be created

queueName

Name of this queue

### **Return Values**

Queue

A Queue with the given name

## **QS\_createReceiver**

### **Purpose**

Creates a QueueReceiver to receive messages from the specified queue.

### **Declaration**

```
QueueReceiver QS_createReceiver(QueueSession session, Queue queue);
```

### **Include**

queue\_session.h

### **Parameters**

session

QueueSession using which the receiver is to be created

queue

The queue to be accessed

### **Return Values**

QueueReceiver

Newly created QueueReceiver, NULL if an error occurs

## **QS\_createReceiverWithSelector**

### **Purpose**

Creates a QueueReceiver to receive messages from the specified queue, using a message selector.

### **Declaration**

```
QueueReceiver QS_createReceiverWithSelector(QueueSession session, Queue queue, mqstring messageSelector);
```

### **Include**

queue\_session.h

### **Parameters**

session

QueueSession using which the receiver is to be created

queue

The queue to be accessed

messageSelector

Only messages with properties matching the message

### **Return Values**

QueueReceiver

Newly created QueueReceiver, NULL if an error occurs

## QS\_createSender

### Purpose

Creates a QueueSender to send messages to the specified queue.

### Declaration

```
QueueSender QS_createSender(QueueSession session, Queue queue);
```

### Include

```
queue_session.h
```

### Parameters

Session

QueueSession using which the sender is to be created

queue

The queue to be accessed, null if this is an unidentified producer

### Return Values

QueueSender

Newly created QueueSender, NULL if an error occurs

## QS\_createStreamMessage

### Purpose

Creates a StreamMessage. A StreamMessage is used to send a self-defining stream of different datatypes.

### Declaration

```
Message QS_createStreamMessage(QueueSession session);
```

### Include

```
queue_session.h
```

### Parameters

session

QueueSession using which the stream message is to be created

**Return Values**

Message

A newly created StreamMessage, NULL if an error occurs

## QS\_createTemporaryQueue

**Purpose**

Creates a temporary queue. Lifetime of the temporary queue is that of the QueueConnection, unless deleted earlier.

**Declaration**

```
TemporaryQueue QS_createTemporaryQueue(QueueSession session);
```

**Include**

queue\_session.h

**Parameters**

session

QueueSession using which the temporary queue is to be created

**Return Values**

TemporaryQueue

A temporary queue identity

## QS\_createTextMessage

**Purpose**

Creates a text message. A TextMessage is used to send a message containing an mqstring.

**Declaration**

```
Message QS_createTextMessage(QueueSession session);
```

**Include**

queue\_session.h

**Parameters**

session

QueueSession using which the text message is to be created

**Return Values**

Message

A newly created TextMessage, NULL if an error occurs

## QS\_createTextMessageWithText

**Purpose**

Creates an initialized text message. A TextMessage is used to send a message containing an mqstring.

**Declaration**

```
Message QS_createTextMessageWithText(QueueSession session, mqstring text);  
queue_session.h
```

**Parameters**

session

QueueSession using which the text message is to be created

text

The string used to initialize this message

**Return Values**

Message

A newly created TextMessage, NULL if an error occurs

## QS\_free

**Purpose**

Frees all the resources allocated on behalf of the QueueSession.

**Declaration**

```
void QS_free(QueueSession session);
```

**Include**

queue\_session.h

**Parameters**

session

QueueSession that has to be freed

## QS\_getMessageListener

### Purpose

Returns the distinguished message listener of the session.

### Declaration

```
MessageListener QS_getMessageListener(QueueSession session);
```

### Include

```
queue_session.h
```

### Parameters

session

QueueSession whose message listener is desired

### Return Values

MessageListener

The message listener associated with this session

Also, refer to the section "QS\_setMessageListener"

## QS\_getTransacted

### Purpose

Indicates if the session is in transacted mode.

### Declaration

```
mqboolean QS_getTransacted(QueueSession session);
```

### Include

```
queue_session.h
```

### Parameters

session

QueueSession that has to be checked

### Return Values

mqboolean

TRUE if in transacted mode



## QS\_recover

### Purpose

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

### Declaration

```
mqboolean QS_recover(QueueSession session);  
queue_session.h
```

### Parameters

session

QueueSession that has to be recovered

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## QS\_rollback

### Purpose

Rollbacks any messages done in this transaction and releases any locks held at the time of call of this API.

### Declaration

```
mqboolean QS_rollback(QueueSession session);
```

### Include

queue\_session.h

### Parameters

session

QueueSession that has to be rolled back

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## QS\_setFioranoMessageListener

### Purpose

Sets the distinguished MessageListener of the session. This API can be used to set a FioranoMessageListener callback pointer that can be passed a parameter, which in turn can be used in the implementation of the callback function.

### Declaration

```
mqboolean QS_setFioranoMessageListener(Queuesession session, FioranoMessageListener ptrmessageListener, void* param);
```

### Include

queue\_session.h

### Parameters

session

QueueSession whose message listener is to be set

ptrmessageListener

Listener to which the messages are to be delivered. This listener accepts a Message structure and a void\* parameter as its arguments.

param

Parameter to be used in the MessageListener callback

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## QS\_setMessageListener

### Purpose

Sets the distinguished message listener of the session.

### Declaration

```
mqboolean QS_setMessageListener(QueueSession session, MessageListener listener);
```

### Include

queue\_session.h

### Parameters

session

QueueSession on which the message listener is to be set

listener

The message listener to associate with this session

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## QSNDR\_close

### Purpose

Closes the queue sender and the resources allocated on behalf of the QueueSender.

### Declaration

```
mqboolean QSNDR_close(QueueSender sender);
```

### Include

queue\_sender.h

### Parameters

sender

QueueSender to be closed

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## QSNDR\_free

### Purpose

Frees the resources allocated on behalf of the queue sender

### Declaration

```
void QSNDR_free(QueueSender sender);
```

### **Include**

queue\_sender.h

### **Parameters**

sender

QueueSender that is to be freed

## **QSNDR\_getDeliveryMode**

### **Purpose**

Gets the default delivery mode of the sender.

### **Declaration**

```
mqint QSNDR_getDeliveryMode(QueueSender sender);
```

### **Include**

queue\_sender.h

### **Parameters**

sender

QueueSender whose delivery mode is desired

### **Return Values**

mqint

The message delivery mode for this queue sender

Also, refer to the section “QSNDR\_setDeliveryMode”

## **QSNDR\_getDisableMessageID**

### **Purpose**

Gets an indication of whether message IDs are disabled.

### **Declaration**

```
mqboolean QSNDR_getDisableMessageID(QueueSender sender);
```

### **Include**

queue\_sender.h

### **Parameters**

sender

QueueSender for which the messageId disable indication is to be checked.

mqboolean

An indication of whether message IDs are disabled

## **QSNDR\_getDisableMessageTimestamp**

### **Purpose**

Gets an indication of whether message timestamps are disabled.

### **Declaration**

```
mqboolean QSNDR_getDisableMessageTimestamp(QueueSender sender);
```

### **Include**

queue\_sender.h

### **Parameters**

sender

QueueSender for which the message timestamp disable indication is to be checked

### **Return Values**

mqboolean

An indication of whether message timestamps are disabled

Also, refer to the section "QSNDR\_setDisableMessageTimestamp"

## **QSNDR\_getPriority**

### **Purpose**

Gets the default priority of the sender.

### **Declaration**

```
mqint QSNDR_getPriority(QueueSender sender);
```

### **Include**

queue\_sender.h

### **Parameter**

sender

QueueSender whose priority is desired

### **Return Values**

mqint

The message priority for this queue sender

Also, refer to the section “QSNDR\_setPriority”

## **QSNDR\_getQueue**

### **Purpose**

Gets the queue associated with this queue sender.

### **Declaration**

```
Queue QSNDR_getQueue(QueueSender sender);
```

### **Include**

queue\_sender.h

### **Parameters**

sender

QueueSender whose queue is desired

### **Return Values**

Queue

The queue for this sender

## QSNDR\_getTimeToLive

### Purpose

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

### Declaration

```
mq_long QSNDR_getTimeToLive(QueueSender sender);
```

### Include

```
queue_sender.h
```

### Parameters

sender

QueueSender whose default message time to live is desired

### Return Values

```
mq_long
```

The message time to live in milliseconds; zero is unlimited

Also, refer to the section “QSNDR\_getTimeToLive”

## QSNDR\_send

### Purpose

Sends a message to the queue on which the sender was created. Uses the default delivery mode, timeToLive and priority of the QueueSender.

### Declaration

```
mq_boolean QSNDR_send(QueueSender sender, Message message);
```

### Include

```
queue_sender.h
```

### Parameters

sender

QueueSender that is used for sending the message

message

The message to be sent

## Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## QSNDR\_send1

### Purpose

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

### Declaration

```
mqboolean QSNDR_send1(QueueSender sender, Message message, mqint deliveryMode, mqint  
priority, mqlong timeToLive);
```

### Include

queue\_sender.h

### Parameters

sender

QueueSender that is used for sending the message

message

The message to be sent

deliveryMode

The delivery mode to be used

priority

The priority for this message

timeToLive

Time-to-live (in milliseconds) of the message

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs



## QSNDR\_send2

### Purpose

Sends a message to the specified queue. Uses the default delivery mode, timeTo-Live and priority of the QueueSender.

### Declaration

```
mqboolean QSNDR_send2(QueueSender sender, Queue queue, Message message);  
queue_sender.h
```

### Parameters

sender

QueueSender that is used for sending the message

queue

The queue to send this message to

message

The message to be sent

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## QSNDR\_send3

### Purpose

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

### Declaration

```
mqboolean QSNDR_send3(QueueSender sender, Queue queue, Message message, mqint  
deliveryMode, mqint priority, mqlong timeToLive);
```

### Include

queue\_sender.h

### Parameters

sender

QueueSender that is used for sending the message

queue

Queue on which this message is to be sent

message

The message to be sent

deliveryMode

The delivery mode to be used The priority for this message

timeToLive

Time-to-live (in milliseconds) of the message

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **QSNDR\_setDeliveryMode**

### **Purpose**

Sets the delivery mode for the sender. By default, delivery mode is set to PERSISTENT.

### **Declaration**

```
mqboolean QSNDR_setDeliveryMode(QueueSender sender, mqint deliveryMode);
```

### **Include**

queue\_sender.h

### **Parameters**

sender

QueueSender whose delivery mode is to be set

deliveryMode

The message delivery mode for this queue sender; legal values are NON\_PERSISTENT and PERSISTENT.

For details, refer to the section "CRTL Constants"

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

Also, refer to the section “[QSNDR\\_getPriority](#)”

## QSNDR\_setDisableMessageID

### Purpose

Sets whether message IDs are disabled.

### Declaration

```
mqboolean QSNDR_setDisableMessageID(QueueSender sender, mqboolean value);
```

### Include

queue\_sender.h

### Parameters

sender

QueueSender for whose messages the messageIDs are to be disabled

value

Indicates if message IDs are disabled

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

Also, refer to the section “[QSNDR\\_getDisableMessageID](#)”

## QSNDR\_setDisableMessageTimestamp

### Purpose

Sets whether message timestamps are disabled.

### Declaration

```
mqboolean QSNDR_setDisableMessageTimestamp(QueueSender sender, mqboolean value);
```

### Include

queue\_sender.h

### Parameter

sender

QueueSender for whose messages the messageIDs are to be disabled

value

Indicates if message timestamps are disabled

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

Also, refer to the section “QSNDR\_getDisableMessageTimestamp”

## **QSNDR\_setPriority**

### **Purpose**

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

### **Declaration**

```
mqboolean QSNDR_setPriority(QueueSender sender, mqint messagePriority);
```

### **Include**

queue\_sender.h

### **Parameters**

sender

QueueSender for whose messages the priority is to be set

messagePriority

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

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **QSNDR\_setTimeToLive**

### **Purpose**

Sets the default length of time in milliseconds from its dispatch time, for which a produced message should be retained by the message system.

By default, Time to live is set to zero.

**Declaration**

```
mqboolean QSNDR_setTimeToLive(QueueSender sender, mqlong timeToLive);
```

**Include**

```
queue_sender.h
```

**Parameters**

sender

QueueSender for whose messages the time to live is to be set

timeToLive

The message time to live in milliseconds; zero is unlimited

**Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## T\_free

**Purpose**

Frees the memory allocated to the topic structure.

**Declaration**

```
void T_free(Topic topic);
```

**Include**

```
topic.h
```

**Parameters**

topic

The topic that has to be freed

## T\_getTopicName

**Purpose**

Returns the topic name as mqstring.

**Declaration**

```
mqstring T_getTopicName(Topic topic);
```

**Include**

topic.h

**Parameters**

topic

Topic whose name is desired

**Return Value**

mqstring

Name of the topic

## TC\_close

**Purpose**

Terminates the connection with Fiorano JMS server and closes the resources allocated on behalf of the TopicConnection.

**Declaration**

```
mqboolean TC_close(TopicConnection tc);
```

**Include**

topic\_connection.h

**Parameters**

tc

TopicConnection that is to be closed

**Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## TC\_createTopicSession

**Purpose**

Creates a TopicSession with the given mode.

**Declaration**

```
TopicSession TC_createTopicSession(TopicConnection tc, mqboolean transacted, mqint acknowledgeMode);
```

### **Include**

topic\_connection.h

### **Parameters**

tc

TopicConnection

mqboolean

Transacted, if TRUE the session is transacted

acknowledgeMode

Indicates whether the consumer or the client acknowledges any messages it receives.

### **Return Values**

TopicSession

A newly created TopicSession, NULL if an error occurs

## **TC\_free**

### **Purpose**

Frees all the resources allocated for the TopicConnection

### **Declaration**

```
void TC_free(TopicConnection tc);
```

### **Include**

topic\_connection.h

### **Parameters**

tc

## **TC\_getClientID**

### **Purpose**

Gets the client identifier for this connection.

### **Declaration**

```
mqstring TC_getClientID(TopicConnection tc);
```

**Include**

topic\_connection.h

**Parameters**

tc

TopicConnection whose clientId is desired

**Return Values**

mqstring

The unique client identifier

## TC\_getExceptionListener

**Purpose**

Gets the ExceptionListener object for this connection. Certain connections may not have an ExceptionListener associated with it.

**Declaration**

```
struct _ExceptionListener* TC_getExceptionListener(TopicConnection tc);
```

**Include**

topic\_connection.h

**Parameters**

connection

TopicConnection whose exception listener is desired

**Return Values**

ExceptionListener

Also, refer to the section "TC\_setExceptionListener"

## TC\_setClientID

**Purpose**

Sets the client identifier for this connection.

**Declaration**

```
mqboolean TC_setClientID(TopicConnection tc, mqstring clientID);
```



**Include**

topic\_connection.h

**Parameters**

connection

TopicConnection

clientID

The unique client identifier

**Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## TC\_setExceptionListener

**Purpose**

Sets an exception listener for this connection. If a JMS provider detects a serious aberration with a connection, it notifies the ExceptionListener of the connection, if one has been registered. It does this by calling the onException method of the listener, sending it the error stack describing the problem.

An exception listener allows a client to be asynchronously notified of a problem. Certain connections only consume messages, hence these connections do not have any other method to be notified of connection failure.

```
mqboolean TC_setExceptionListener(TopicConnection connection, ExceptionListener-Pointer ptrListener, void* param);
```

**Include**

topic\_connection.h

**Parameters**

connection

TopicConnection for which the exception listener is to be registered

ptrListener

Function pointer for callback method of the exception listener

param

Any parameter that is to be used in the callback method

**Return Values**

mqboolean

TRUE if successful, false if an error occurs

## TC\_start

**Purpose**

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

**Declaration**

```
mqboolean TC_start(TopicConnection tc);
```

**Include**

topic\_connection.h

**Parameters**

tc

TopicConnection that is to be started

**Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## TC\_stop

**Purpose**

Temporarily stops delivery of incoming messages of a connection. Delivery can be restarted using the start method of a connection. When the connection is stopped, delivery to all the message consumers of the connection is inhibited: synchronous receives block, and messages are not delivered to message listeners.

**Declaration**

```
mqboolean TC_stop(TopicConnection tc);
```

**Include**

topic\_connection.h

### Parameters

tc

TopicConnection that is to be stopped

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## TCF\_createTopicConnection

### Purpose

Creates a queue connection with specified user identity.

### Declaration

```
TopicConnection TCF_createTopicConnection(TopicConnectionFactory tcf, mqstring  
username, mqstring password);
```

### Include

topic\_connection\_factory.h

### Parameters

tcf

TopicConnectionFactory to be used to create the connection

username

The user name to be used to create the connection The password for the specified username

### Return Values

TopicConnection

A newly created TopicConnection, NULL if an error occurs

## TCF\_createTopicConnectionDefParams

### Purpose

Creates a queue connection with default user identity.

### Declaration

```
TopicConnection TCF_createTopicConnectionDefParams(TopicConnectionFactory tcf);
```

### **Include**

topic\_connection\_factory.h

### **Parameters**

tcf

TopicConnectionFactory to be used to create the connection

### **Return Values**

TopicConnection

A newly created TopicConnection, NULL if an error occurs

## **TCF\_free**

### **Purpose**

Frees all the resources allocated for the TopicConnectionFactory.

### **Declaration**

```
void TCF_free(TopicConnectionFactory tcf);
```

### **Include**

topic\_connection\_factory.h

### **Parameters**

tcf

TopicConnectionFactory that is to be freed

## **TMPQ\_delete**

### **Purpose**

Deletes this temporary queue. If there are existing receivers still using it, an error occurs.

### **Declaration**

```
mqboolean TMPQ_delete(TemporaryQueue tempQueue);
```

### **Include**

temporary\_queue.h

### Parameters

tempQueue

The temporary queue to be deleted

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## TMPQ\_free

### Purpose

Frees all the resources allocated on behalf of a temporary queue.

### Declaration

```
void TMPQ_free(TemporaryQueue tempQueue);
```

### Include

temporary\_queue.h

### Parameters

tempQueue

TemporaryQueue to be freed

## TMPT\_delete

### Purpose

Deletes this temporary topic. If there are existing publishers or subscribers still using it, an error occurs.

### Declaration

```
mqboolean TMPT_delete(TemporaryTopic tempTopic);
```

### Include

temporary\_topic.h

### Parameters

tempTopic

The temporary topic to be deleted

## Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## TMPT\_free

### Purpose

Frees all the resources allocated on behalf of the temporary topic.

### Declaration

```
void TMPT_free(TemporaryTopic tempTopic);
```

### Include

temporary\_topic.h

### Parameters

tempTopic

The TemporaryTopic to be freed

## TP\_close

### Purpose

Closes the TopicPublisher and the resources allocated on behalf of the TopicPublisher.

### Declaration

```
mqboolean TP_close(TopicPublisher tp);
```

### Include

topic\_publisher.h

### Parameters

tp

TopicPublisher that is to be closed

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## TP\_free

### Purpose

Frees the resources allocated on behalf of the topic publisher.

### Declaration

```
void TP_free(TopicPublisher pub);
```

### Include

topic\_publisher.h

### Parameters

pub

TopicPublisher that is to be freed

## TP\_getDeliveryMode

### Purpose

Gets the default delivery mode of the producer.

### Declaration

```
mqint TP_getDeliveryMode(TopicPublisher tp);
```

### Include

topic\_publisher.h

### Parameters

tp

TopicPublisher whose delivery mode is desired

### Return Values

mqint

The message delivery mode for this message producer

## TP\_getDisableMessageID

### Purpose

Gets an indication of whether message IDs are disabled.

### Declaration

```
mqboolean TP_getDisableMessageID(TopicPublisher tp);
```

### Include

```
topic_publisher.h
```

### Parameters

tp

TopicPublisher for which the messageID disable indication has to be checked

### Return Values

mqboolean

An indication of whether message IDs are disabled

## TP\_getDisableMessageTimestamp

### Purpose

Gets an indication of whether message timestamps are disabled.

### Declaration

```
mqboolean TP_getDisableMessageTimestamp(TopicPublisher tp);
```

### Include

```
topic_publisher.h
```

### Parameters

tp

TopicPublisher for which the message timestamp indication has to be checked

### Return Values

mqboolean

An indication of whether message IDs are disabled



## TP\_getPriority

### Purpose

Gets the default priority of the producer.

### Declaration

```
mqint TP_getPriority(TopicPublisher tp);
```

### Include

```
topic_publisher.h
```

### Parameters

tp

TopicPublisher whose priority is desired

### Return Values

mqint

The message priority for this message producer

## TP\_getTimeToLive

### Purpose

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

### Declaration

```
mqlong TP_getTimeToLive(TopicPublisher tp);
```

### Include

```
topic_publisher.h
```

### Parameters

tp

TopicPublisher whose message time to live is desired

### Return Values

mqlong

The message time to live in milliseconds; zero is unlimited

## TP\_getTopic

### Purpose

Gets the topic associated with this publisher.

### Declaration

```
Topic TP_getTopic(TopicPublisher pub);
```

### Include

```
topic_publisher.h
```

### Parameters

pub

TopicPublisher whose topic is desired

### Return Values

Topic

Topic associated with the publisher

## TP\_publish

### Purpose

Publishes a message to the topic on which the publisher is created. Uses the default delivery mode, timeToLive and priority of the topic.

### Declaration

```
mqboolean TP_publish(TopicPublisher pub, Message message);
```

### Include

```
topic_publisher.h
```

### Parameters

pub

TopicPublisher that is to be used to publish the message

message

The message to be published

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## TP\_publish1

### Purpose

Publishes a message to the topic on which the publisher is created, specifying delivery mode, priority and time to live to the topic.

### Declaration

```
mqboolean TP_publish1(TopicPublisher pub, Message message, mqint deliveryMode, mqint priority, mqlong timeToLive);
```

### Include

```
topic_publisher.h
```

### Parameters

pub

TopicPublisher to be used for publishing the message

message

deliveryMode

The delivery mode to use

priority

Priority for this message

timeToLive

Lifetime (in milliseconds) of the message

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## TP\_publish2

### Purpose

Publishes a message to a topic for an unidentified message producer. Uses the default delivery mode, timeToLive and priority of the topic.

### Declaration

```
mqboolean TP_publish2(TopicPublisher pub, Topic topic, Message message);
```

### Include

```
topic_publisher.h
```

### Parameters

pub

TopicPublisher on which message is to be published

topic

The topic to publish this message to

message

The message to be published

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## TP\_publish3

### Purpose

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

### Declaration

```
mqboolean TP_publish3(TopicPublisher pub, Topic topic, Message message, mqint  
deliveryMode, mqint priority, mqlong timeToLive);
```

### Include

```
topic_publisher.h
```

### Parameters

pub

TopicPublisher on which the message is to be published

topic

The topic to publish this message to

message

The message to be published

deliveryMode

The delivery mode to use

priority

The priority for this message

timeToLive

The lifetime (in milliseconds) of the message

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **TP\_setDeliveryMode**

### **Purpose**

Sets the default delivery mode of the producer.

### **Declaration**

```
mqboolean TP_setDeliveryMode(TopicPublisher tp, mqint deliveryMode);
```

### **Include**

topic\_publisher.h

### **Parameters**

tp

TopicPublisher for whose messages the delivery mode is to be set.

deliveryMode

The message delivery mode for this message producer; legal values are NON\_PERSISTENT and PERSISTENT.

For more information, read the CRTL Constants section

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **TP\_setDisableMessageID**

### **Purpose**

Sets whether message IDs are disabled for this Topic Publisher.

### **Declaration**

```
mqboolean TP_setDisableMessageID(TopicPublisher tp, mqboolean value);
```

### **Include**

topic\_publisher.h

### **Parameters**

tp

TopicPublisher for which the messageID is to be disabled

value

Indicates if message IDs are disabled

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **TP\_setDisableMessageTimestamp**

### **Purpose**

Sets whether message timestamps are disabled.

### **Declaration**

```
mqboolean TP_setDisableMessageTimestamp(TopicPublisher tp, mqboolean value);
```

### **Include**

topic\_publisher.h

### **Parameters**

tp

TopicPublisher for which the message timestamp is to be disabled  
value

Indicates if message timestamps are disabled

**Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## TP\_setPriority

**Purpose**

Sets the default priority of the producer.

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

**Declaration**

```
mqboolean TP_setPriority(TopicPublisher tp, mqint msgPriority);
```

**Include**

topic\_publisher.h

**Parameters**

tp

TopicPublisher for which the default priority is to be set

msgPriority

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

**Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## TP\_setTimeToLive

### Purpose

Sets the default length of time in milliseconds, from its dispatch time, for which a produced message should be retained by the message system.

By default, Time to live is set to zero.

### Declaration

```
mqboolean TP_setTimeToLive(TopicPublisher tp, mqlong timeToLive);
```

### Include

```
topic_publisher.h
```

### Parameters

tp

TopicPublisher for which the time to live is to be set

timeToLive

The message time to live in milliseconds; zero is unlimited

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## TRQST\_close

### Purpose

Closes the topic requestor and all the resources allocated on behalf of TopicRequestor.

### Declaration

```
mqboolean TRQST_close(TopicRequestor requestor);
```

### Include

```
topic_requestor.h
```

### Parameters

requestor

TopicRequestor that is to be closed



## Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## TRQST\_free

### Purpose

Frees the resources allocated on behalf of TopicRequestor.

### Declaration

```
void TRQST_free(TopicRequestor requestor);
```

### Include

topic\_requestor.h

### Parameters

requestor

TopicRequestor that is to be freed

## TRQST\_request

### Purpose

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

```
Message TRQST_request(TopicRequestor requestor, Message message);
```

### Include

topic\_requestor.h

### Parameters

requestor

TopicRequestor that is to be used for sending the requests

message

The request to send

**Return Values**

Message

The reply message, NULL if an error occurs

## TRQST\_requestWithTimeout

**Purpose**

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

**Declaration**

```
Message TRQST_requestWithTimeout(TopicRequestor requestor, Message message, mqlong timeout);
```

**Include**

```
topic_requestor.h
```

**Parameters**

requestor

TopicRequestor that is to be used for sending the requests

message

The request to send

timeout

The time for which the requestor waits for a reply

**Return Values**

Message

The reply message, NULL if an error occurs

## TS\_close

**Purpose**

Closes the topic session and resources allocated on behalf of the TopicSession.

**Declaration**

```
mqboolean TS_close(TopicSession ts);
```

### **Include**

topic\_session.h

### **Parameters**

ts

TopicSession to be closed

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **TS\_commit**

### **Purpose**

Commits all messages done in this transaction and releases any locks held at the time of call of this API.

### **Declaration**

```
mqboolean TS_commit(TopicSession ts);
```

### **Include**

topic\_session.h

### **Parameters**

ts

TopicSession to be committed

### **Return Values**

TRUE if successful, FALSE if an error occurs

## **TS\_createBytesMessage**

### **Purpose**

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

### **Declaration**

```
Message TS_createBytesMessage(TopicSession ts);
```

**Include**

topic\_session.h

**Parameters**

ts

TopicSession using which the bytes message is to be created

**Return Values**

Message

Newly created BytesMessage, NULL if an error occurs

## TS\_createDurableSubscriber

**Purpose**

Creates a durable Subscriber to the specified topic.

If a client needs to receive all the messages published on a topic, including the ones published while the subscriber is inactive, it uses a durable TopicSubscriber. The FioranoMQ server retains a record of this durable subscription and insures that all messages from the publishers of the topic are retained, until they are acknowledged by this durable subscriber or have expired.

Sessions with durable subscribers must always provide the same client identifier. In addition, each client must specify a name that uniquely identifies (within client identifier) each durable subscription it creates. Only one session at a time can have a TopicSubscriber for a particular durable subscription.

A client can change an existing durable subscription by creating a durable Topic-Subscriber with the same name and a new topic and/or message selector. Changing a durable subscriber is equivalent to unsubscribing (deleting) the old one and creating a new one.

**Declaration**

```
TopicSubscriber TS_createDurableSubscriber(TopicSession ts, Topic topic, mqstring name);
```

**Include**

topic\_session.h

**Parameters**

ts

TopicSession using which the durable subscriber is to be created

topic

The non-temporary topic to subscribe to

name

The name used to identify this subscription

### **Return Values**

TopicSubscriber

Newly created TopicSubscriber, NULL if an error occurs

## **TS\_createDurableSubscriberWithSelector**

### **Purpose**

Creates a Durable Subscriber to the given Topic with given message selector and local message delivery options.

If a client needs to receive all the messages published on a topic, including the ones published while the subscriber is inactive, it uses a durable TopicSubscriber. The FioranoMQ server retains a record of this durable subscription and insures that all messages from the publishers of the topic are retained until they are acknowledged by this durable subscriber or they have expired.

Sessions with durable subscribers must always provide the same client identifier. In addition, each client must specify a name that uniquely identifies (within client identifier) each durable subscription it creates. Only one session at a time can have a TopicSubscriber for a particular durable subscription.

A client can change an existing durable subscription by creating a durable Topic-Subscriber with the same name and a new topic and/or message selector. Changing a durable subscriber is equivalent to unsubscribing (deleting) the old one and creating a new one.

In certain instances, a connection may both publish and subscribe to a topic. The subscriber NoLocal attribute allows a subscriber to inhibit the delivery of messages published by its own connection. The default value for this attribute is false.

### **Declaration**

```
TopicSubscriber TS_createDurableSubscriberWithSelector(TopicSession ts, Topic topic,
mqstring subscriberID, mqstring messageselector, mqboolean nlocal);
```

### **Include**

topic\_session.h

### **Parameters**

ts

TopicSession using which the durable subscriber is to be created

topic

The non-temporary topic to subscribe to

subscriberID

The name used to identify this subscription

messageselector

Only messages with properties matching the message selector expression are delivered

noLocal

If set, inhibits the delivery of messages published on the same connection

### **Return Values**

TopicSubscriber

Newly created TopicSubscriber, NULL if an error occurs

## **TS\_createMapMessage**

### **Purpose**

Creates a map message. A MapMessage object is used to send a self-defining set of name-value pairs, where names are of type mqstring and values can be of different datatypes.

### **Declaration**

```
Message TS_createMapMessage(TopicSession ts);
```

### **Include**

topic\_session.h

### **Parameters**

ts

TopicSession using which the map message is to be created

### **Return Values**

Message

Newly created MapMessage, NULL if an error occurs

## **TS\_createPublisher**

### **Purpose**

Creates a Publisher for the specified topic.

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

### **Declaration**

```
TopicPublisher TS_createPublisher(TopicSession ts, Topic topic);
```

### **Include**

```
topic_session.h
```

### **Parameters**

ts

TopicSession using which the publisher is to be created

topic

The topic to publish to, null if this is an unidentified producer

### **Return Values**

TopicPublisher

Newly created TopicPublisher, NULL if an error occurs

## **TS\_createPublisherOnTempTopic**

### **Purpose**

Creates a Publisher for the specified temporary topic.

### **Declaration**

```
TopicPublisher TS_createPublisherOnTempTopic(TopicSession ts, TemporaryTopic temptopic);
```

### **Include**

```
topic_session.h
```

### **Parameters**

ts

TopicSession using which the publisher is to be created

temptopic

The topic to publish to, null if this is an unidentified producer

### **Return Values**

TopicPublisher

Newly created TopicPublisher, NULL if an error occurs

## TS\_createStreamMessage

### Purpose

Creates a StreamMessage. A StreamMessage is used to send a self-defining stream of different datatypes.

### Declaration

```
Message TS_createStreamMessage(TopicSession ts);
```

### Include

```
topic_session.h
```

### Parameters

ts

TopicSession using which the stream message is to be created

Message

Newly created StreamMessage, NULL if an error occurs

## TS\_createSubOnTempTopicWS

### Purpose

Creates a non-durable Subscriber to the specified temporary topic using the specified message selector.

### Declaration

```
TopicSubscriber TS_createSubOnTempTopicWS(TopicSession ts, TemporaryTopic temptopic,  
mqstring messageSelector, mqboolean nolocal);
```

### Include

```
topic_session.h
```

### Parameters

ts

TopicSession using which the subscriber is to be created

temptopic

The temporarytopic to subscribe to

messageselector

Only messages with properties matching the message selector expression are delivered



noLocal

If set, inhibits the delivery of messages published on the same connection

### Return Values

TopicSubscriber

Newly created TopicSubscriber, NULL if an error occurs

## TS\_createSubscriber

### Purpose

Creates a non-durable Subscriber to the specified topic. A client uses a TopicSubscriber to receive messages that have been published to a topic.

### Declaration

```
TopicSubscriber TS_createSubscriber(TopicSession ts, Topic topic);
```

### Include

topic\_session.h

### Parameters

ts

TopicSession using which the subscriber is to be created

topic

The topic to subscribe to

### Return Values

TopicSubscriber

Newly created TopicSubscriber, NULL if an error occurs

## TS\_createSubscriberOnTempTopic

### Purpose

Creates a non-durable Subscriber to the specified temporary topic.

### Declaration

```
TopicSubscriber TS_createSubscriberOnTempTopic(TopicSession ts, TemporaryTopic temptopic);
```

**Include**

topic\_session.h

**Parameters**

ts

TopicSession using which the subscriber is to be created

temptopic

The temporary topic to subscribe to

**Return Values**

TopicSubscriber

## TS\_createSubscriberWithSelector

**Purpose**

Creates a non-durable Subscriber to the specified topic using the specified message selector.

A client uses a TopicSubscriber to receive messages that have been published to a topic. Regular TopicSubscribers are not durable. They receive only messages that are published while they are active.

Messages filtered out by the message selector of a subscriber would never be delivered to the subscriber. From the perspective of a subscriber, they do not exist.

In certain instances, a connection may both publish and subscribe to a topic. The subscriber NoLocal attribute allows a subscriber to inhibit the delivery of messages published by its own connection. The default value for this attribute is false.

**Declaration**

```
TopicSubscriber TS_createSubscriberWithSelector(TopicSession ts, Topic topic, mqstring messageSelector, mqboolean noLocal);
```

**Include**

topic\_session.h

**Parameters**

ts

TopicSession using which the subscriber is to be created

topic

The topic to subscribe to

messageselector

Only messages with properties matching the message selector

`noLocal`

If set, inhibits the delivery of messages published on the same connection

### **Return Values**

`TopicSubscriber`

Newly created `TopicSubscriber`, NULL if an error occurs

## **TS\_createTemporaryTopic**

### **Purpose**

Creates a temporary topic. Lifetime of the topic is that of the `TopicConnection`, unless deleted earlier.

### **Declaration**

```
TemporaryTopic TS_createTemporaryTopic(TopicSession ts);
```

### **Include**

`topic_session.h`

### **Parameters**

`ts`

`TopicSession` using which the temporary topic is to be created

### **Return Values**

`TemporaryTopic`

A temporary topic identity; NULL if an error occurs

## **TS\_createTextMessage**

### **Purpose**

Creates a text message. A `TextMessage` is used to send a message containing an `mqstring`.

### **Declaration**

```
Message TS_createTextMessage(TopicSession ts);
```

### **Include**

`topic_session.h`

### Parameters

ts

TopicSession using which the text message is to be created

### Return Values

Message

Newly created TextMessage, NULL if an error occurs

## TS\_createTextMessageWithText

### Purpose

Creates an initialized text message. A TextMessage is used to send a message containing an mqstring.

### Declaration

```
Message TS_createTextMessageWithText(TopicSession ts, mqstring text);
```

### Include

topic\_session.h

### Parameters

ts

TopicSession using which the text message is to be created

text

The string used to initialize this message

### Return Values

Message

Newly created TextMessage, NULL if an error occurs

## TS\_createTopic

### Purpose

Creates a topic identity given a topic name.

### Declaration

```
Topic TS_createTopic(TopicSession ts, mqstring topicname);
```

### Include

topic\_session.h

### Parameters

ts

TopicSession using which the topic is to be created

topicname

### Return Values

Topic

A Topic with the given name; NULL if an error occurs

## TS\_free

### Purpose

Frees all the resources allocated on behalf of the TopicSession.

### Declaration

```
void TS_free(TopicSession ts);
```

### Include

topic\_session.h

### Parameters

ts

TopicSession to be freed

## TS\_getMessageListener

### Purpose

Returns the distinguished message listener of the session.

### Declaration

```
MessageListener TS_getMessageListener(TopicSession ts);
```

### Include

topic\_session.h

### Parameters

ts

TopicSession whose message listener is desired

### Return Values

MessageListener

## TS\_getTransacted

### Purpose

Gets an indication whether the session is transacted.

### Declaration

```
mqboolean TS_getTransacted(TopicSession ts, mqboolean *trans);
```

### Include

topic\_session.h

### Parameters

ts

TopicSession that is to be checked

trans

Pointer to the boolean in which the return value is to be set

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## TS\_recover

### Purpose

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

### Declaration

```
mqboolean TS_recover(TopicSession ts);
```

### Include

topic\_session.h

### Parameters

ts

TopicSession that is to be recovered

**Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## TS\_rollback

**Purpose**

Rollbacks any messages done in this transaction and releases any locks held at the time of call of this API.

**Declaration**

```
mqboolean TS_rollback(TopicSession ts);
```

**Include**

topic\_session.h

**Parameters**

ts

TopicSession to be rolledback

**Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## TS\_setFioranoMessageListener

**Purpose**

Sets the distinguished MessageListener of the session. This API can be used to set a FioranoMessageListener callback pointer that can be passed a parameter, which in turn can be used in the implementation of the callback function.

**Declaration**

```
mqboolean TS_setFioranoMessageListener(TopicSession session, FioranoMessageListener ptrmessageListener, void* param);
```

**Include**

topic\_session.h

### Parameters

session

TopicSession whose message listener is to be set

ptrmessageListener

Listener to which the messages are to be delivered. This listener accepts a Message structure and a void\* parameter as its arguments.

param

Parameter to be used in the MessageListener callback

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## TS\_setMessageListener

### Purpose

Sets the distinguished message listener of the session.

### Declaration

```
mqboolean TS_setMessageListener(TopicSession ts, MessageListener listener);
```

### Include

topic\_session.h

### Parameters

ts

TopicSession whose message listener is to be set

listener

Message listener to associate with this session

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs



## TS\_unsubscribe

### Purpose

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

This method deletes the state being maintained on behalf of the subscriber by the FioranoMQ server.

It is erroneous for a client to delete a durable subscription while there is an active TopicSubscriber for the subscription, while a consumed message is part of a pending transaction, or has not been acknowledged in the session.

### Declaration

```
mqboolean TS_unsubscribe(TopicSession ts, mqstring name);
```

### Include

```
topic_session.h
```

### Parameters

ts

TopicSession using which the subscription is to be unsubscribed

name

The name used to identify this subscription

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## TSUB\_close

### Purpose

Closes the TopicSubscriber and the resources allocated on behalf of TopicSubscriber.

### Declaration

```
mqboolean TSUB_close(TopicSubscriber consumer);
```

### Include

```
topic_subscriber.h
```

consumer

TopicSubscriber that is to be closed

## Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## TSUB\_free

### Purpose

Frees the resources allocated on behalf of TopicSubscriber.

### Declaration

```
void TSUB_free(TopicSubscriber consumer);
```

### Include

topic\_subscriber.h

### Parameters

consumer

TopicSubscriber that is to be freed

## TSUB\_getMessageListener

### Purpose

Gets the MessageListener of this message consumer.

### Declaration

```
MessageListener TSUB_getMessageListener(TopicSubscriber consumer);
```

### Include

topic\_subscriber.h

### Parameters

consumer

TopicSubscriber whose message listener is desired

MessageListener

Listener for the message consumer, NULL if there is none set

## TSUB\_getMessageSelector

### Purpose

Gets the message selector expression of this message consumer.

### Declaration

```
mqstring TSUB_getMessageSelector(TopicSubscriber consumer);
```

### Include

topic\_subscriber.h

### Parameters

consumer

TopicSubscriber whose message selector is desired

### Return Values

mqstring

Message selector of this message consumer

## TSUB\_getNoLocal

### Purpose

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

### Declaration

```
mqboolean TSUB_getNoLocal(TopicSubscriber consumer);
```

### Include

topic\_subscriber.h

### Parameters

consumer

TopicSubscriber whose noLocal attribute is desired

mqboolean

TRUE if locally published messages are being inhibited

## TSUB\_getTopic

### Purpose

Gets the topic associated with this subscriber.

### Declaration

```
Topic TSUB_getTopic(TopicSubscriber consumer);
```

### Include

```
topic_subscriber.h
```

### Parameters

consumer

TopicSubscriber whose topic is desired

### Return Values

Topic

Topic associated with this subscriber, NULL if an error occurs

## TSUB\_receive

### Purpose

Receives the next message produced for this topic subscriber. This call blocks indefinitely until a message is produced or this topic subscriber is closed.

If this receive is done within a transaction, the subscriber retains the message until the transaction commits.

### Declaration

```
Message TSUB_receive(TopicSubscriber consumer);
```

### Include

```
topic_subscriber.h
```

consumer

TopicSubscriber on which receive is to be called

### Return Values

Message

The next message produced for this message consumer, NULL if this topic subscriber is concurrently closed.

## TSUB\_receiveNoWait

### Purpose

Receives the next message if one is immediately available.

### Declaration

```
Message TSUB_receiveNoWait(TopicSubscriber consumer);
```

### Include

```
topic_subscriber.h
```

### Parameters

consumer

TopicSubscriber on which receive is called

### Return Values

Message

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

## TSUB\_receiveWithTimeout

### Purpose

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

### Declaration

```
Message TSUB_receiveWithTimeout(TopicSubscriber consumer, mqlong timeout);  
topic_subscriber.h
```

### Parameters

consumer

TopicSubscriber on which receive is called.

timeout

The timeout value (in milliseconds)

### Return Values

Message

The next message produced for this message consumer, else NULL is returned

## TSUB\_setFioranoMessageListener

### Purpose

Sets MessageListener of the message consumer. This API can be used to set a FioranoMessageListener callback pointer that can be passed a parameter, which in turn can be used in the implementation of the callback function.

### Declaration

```
mqboolean TSUB_setFioranoMessageListener(TopicSubscriber consumer,  
FioranoMessageListener ptrmessageListener, void* param);
```

### Include

```
topic_subscriber.h
```

### Parameters

consumer

TopicSubscriber whose message listener is to be set

ptrmessageListener

Listener to which the messages are to be delivered. This listener accepts a Message structure and a void\* parameter as its arguments.

param

Parameter to be used in the MessageListener callback

### Return Values

mqboolean

## TSUB\_setMessageListener

### Purpose

Sets the MessageListener of the message consumer.

### Declaration

```
mqboolean TSUB_setMessageListener(TopicSubscriber consumer, MessageListener listener);
```

### Include

```
topic_subscriber.h
```

### Parameters

consumer

TopicSubscriber whose message listener is to be set  
listener

The messages are delivered to this listener

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **UCF\_createConnectionWithParams**

### **Purpose**

Creates a Unified connection with the connection factory based on the lazy thread creation flag

### **Declaration**

```
FioranoConnection UCF_createConnectionWithParams (ConnectionFactory cf, mqstring  
userName, mqstring passWord)
```

### **Include**

unified\_connection\_factory.h

### **Parameters**

cf

ConnectionFactory to be used to create the connection.

### **userName**

The user name to be used to create the connection

### **passWord**

The password for the specified username

### **Return Value**

FioranoConnection

A newly created FioranoConnection, NULL if an error occurs

## UCF\_createConnection

### Purpose

Gets a Connection object from the connection factory.

### Declaration

```
FioranoConnection UCF_createConnectionWithParams (ConnectionFactory cf)
```

### Include

```
unified_connection_factory.h
```

### Parameters

cf

ConnectionFactory to be used to create the connection.

### Return Value

FioranoConnection

A newly created FioranoConnection, NULL if an error occurs

## CF\_free

### Purpose

Frees Connection Factory Object.

### Declaration

```
void CF_free(ConnectionFactory cf)
```

### Include

```
unified_connection_factory.h
```

### Parameters

cf

ConnectionFactory to be freed.



## FC\_createSession

### Purpose

Creates a FioranoSession with the given mode.

### Declaration

```
FioranoSession FC_createSession(FioranoConnection conn, mqboolean trasacted, mqint ackMode)
```

### Include

```
fiorano_connection.h
```

### Parameters

conn

FioranoConnection

trasacted

Transacted, if TRUE the session is transacted

ackMode

Indicates whether the consumer or the client acknowledges any messages it receives.

### Return Values

FioranoSession

A newly created FioranoSession, NULL if an error occurs

## FC\_getClientID

### Purpose

Gets the client identifier for this connection.

### Declaration

```
mqstring FC_getClientID(FioranoConnection conn)
```

### Include

```
fiorano_connection.h
```

### Parameters

conn

FioranoConnection whose clientId is desired

## Return Values

mqstring

The unique client identifier

## FC\_getExceptionListener

### Purpose

Gets the ExceptionListener object for this connection. Certain connections may not have an ExceptionListener associated with it.

### Declaration

```
ExceptionListener FC_getExceptionListener(FioranoConnection conn)
```

### Include

fiorano\_connection.h

### Parameters

conn

FioranoConnection whose exception listener is desired

### Return Values

ExceptionListener

Also, refer to the section "FC\_setExceptionListener"

## FC\_setClientID

### Purpose

Sets the client identifier for this connection.

### Declaration

```
void FC_setClientID(FioranoConnection conn, mqstring clientID)
```

### Include

fiorano\_connection.h

### Parameters

conn

FioranoConnection

clientID

The unique client identifier

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## FC\_setExceptionListener

### Purpose

Sets an exception listener for this connection. If a JMS provider detects a serious aberration with a connection, it notifies the ExceptionListener of the connection, if one has been registered. It does this by calling the onException method of the listener, sending it the error stack describing the problem.

An exception listener allows a client to be asynchronously notified of a problem. Certain connections only consume messages, hence these connections do not have any other method to be notified of connection failure.

### Declaration

```
void FC_setExceptionListener(FioranoConnection conn, ExceptionListener listener);
```

### Include

fiorano\_connection.h

### Parameters

conn

FioranoConnection for which the exception listener is to be registered

listener

Function pointer for callback method of the exception listener

### Return Values

mqboolean

TRUE if successful, false if an error occurs

## FC\_start

### Purpose

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

### Declaration

```
FC_start(FioranoConnection fc)
```

### Include

```
fiorano_connection.h
```

### Parameters

fc

FioranoConnection that is to be started

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## FC\_stop

### Purpose

Temporarily stops delivery of incoming messages of a connection. Delivery can be restarted using the start method of a connection. When the connection is stopped, delivery to all the message consumers of the connection is inhibited: synchronous receives block, and messages are not delivered to message listeners.

### Declaration

```
Void FC_stop(FioranoConnection fc)
```

### Include

```
fiorano_connection.h
```

### Parameters

fc

FioranoConnection that is to be stopped

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## FC\_free

### Purpose

Frees all the resources allocated for the FioranoConnection

### Declaration

```
void FC_free(FioranoConnection fc)
```

### Include

fiorano\_connection.h

### Parameters

fc

FioranoConnection to be freed

## FC\_setUnifiedConnectionID

### Purpose

Sets the client identifier for this unified connection.

### Declaration

```
void FC_setUnifiedConnectionID(FioranoConnection fc, mqstring ucId)
```

### Include

fiorano\_connection.h

### Parameters

fc

FioranoConnection

ucId

The unique client identifier for the unified connection

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## FC\_getUnifiedConnectionID

### Purpose

Gets the client identifier for this unified connection.

### Declaration

```
mqstring FC_getUnifiedConnectionID(FioranoConnection fc)
```

### Include

```
fiorano_connection.h
```

### Parameters

fc

FioranoConnection whose clientId is desired

### Return Values

mqstring

The unique client identifier for the unified Connection

## US\_getMessageListener

### Purpose

Gets the message listener of the session.

### Declaration

```
MessageListener US_getMessageListener(FioranoSession session)
```

### Include

```
unified_session.h
```

### Parameters

session

FioranoSession for which MessageListener to be returned.

### Return Value

MessageListener

A MessageListener Object for the session

## US\_getTransacted

### Purpose

Returns the Boolean whether the session is transacted.

### Declaration

```
mqboolean US_getTransacted(FioranoSession session)
```

### Include

```
unified_session.h
```

### Parameters

session

FioranoSession that is to be checked.

### Return Value

mqboolean

TRUE if session is transacted, FALSE if it is not.

## US\_get AcknowledgeMode

### Purpose

Returns the mode of Acknowledgement.

### Declaration

```
mqint US_getAcknowledgeMode (FioranoSession session)
```

### Include

```
unified_session.h
```

### Parameters

session

FioranoSession for which Acknowledgement mode is to be returned.

### Return Value

mqint

1 if acknowledge mode is AUTO\_ACKNOWLEDGE, 2 if CLIENT\_ACKNOWLEDGE, 3 if DUPS\_OK\_ACKNOWLEDGE

## US\_setMessageListener

### Purpose

Sets a MessageListener to the Session.

### Declaration

```
mqboolean US_setMessageListener (FioranoSession session, MessageListener listener)
```

### Include

```
unified_session.h
```

### Parameters

session

FioranoSession for which MessageListener is to be set.

listener

Message listener to associate with this session

### Return Value

mqboolean

TRUE if successful, FALSE if an error occurs.

## US\_setFioranoMessageListener

### Purpose

Sets a MessageListener to the Session.

### Declaration

```
mqboolean US_setFioranoMessageListener (FioranoSession session, MessageListener listener, void* param)
```

### Include

```
unified_session.h
```

### Parameters

session

FioranoSession for which MessageListener is to be set.

listener



Listener to which the messages are to be delivered. This listener accepts a Message structure and a void\* parameter as its arguments.

param

Parameter to be used in the MessageListener callback

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **US\_createBrowser**

### **Purpose**

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

### **Declaration**

```
QueueBrowser US_createBrowser(FioranoSession session, Queue queue);
```

### **Include**

unified\_session.h

### **Parameters**

session

FioranoSession using which the browser is to be created

queue

The queue to be accessed

### **Return Values**

QueueBrowser

Newly created QueueBrowser, NULL if an error occurs

## US\_createBrowserWithSelector

### Purpose

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

### Declaration

```
QueueBrowser US_createBrowserWithSelector(FioranoSession session, Queue queue,  
mqstring messageSelector);
```

### Include

unified\_session.h

### Parameters

session

FioranoSession using which the browser is to be created

queue

The queue to be accessed

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 browser.

### Return Values

QueueBrowser

Newly created QueueBrowser, NULL if an error occurs

## US\_createConsumer

### Purpose

Creates a FioranoMessageConsumer to receive messages from the specified destination.

### Declaration

```
FioranoMessageConsumer US_createConsumer(FioranoSession session, Destination  
destination)
```

### Include

unified\_session.h

### Parameters

session

FioranoSession using which the Consumer is to be created

destination

The destination to be accessed

### Return Values

FioranoMessageConsumer

Newly created FioranoMessageConsumer, NULL if an error occurs

## US\_createConsumerWithSelector

### Purpose

Creates a FioranoMessageConsumer to receive messages from the specified destination, using a message selector.

### Declaration

```
FioranoMessageConsumer US_createConsumerWithSelector(FioranoSession session,  
Destination destination, mqstring messageSelector, mqboolean nlocal);
```

### Include

unified\_session.h

### Parameters

session

FioranoSession using which the receiver is to be created

destination

The destination to be accessed

messageSelector

Only messages with properties matching the message

nlocal

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

### Return Values

FioranoMessageConsumer

Newly created FioranoMessageConsumer, NULL if an error occurs

## US\_createProducer

### Purpose

Creates a FioranoMessageProducer to send messages to the specified destination.

### Declaration

```
FioranoMessageProducer US_createProducer(FioranoSession session, Destination destination);
```

### Include

unified\_session.h

### Parameters

session

FioranoSession using which the producer is to be created

destination

The destination to be accessed, null if this is an unidentified producer

### Return Values

FioranoMessageProducer

Newly created FioranoMessageProducer, NULL if an error occurs

## US\_createDurableSubscriber

### Purpose

Creates a durable Subscriber to the specified topic.

### Declaration

```
TopicSubscriber US_createDurableSubscriber(FioranoSession session, Topic topic, mqstring name);
```

### Include

unified\_session.h

### Parameters

session

FioranoSession using which the durable subscriber is to be created

topic

The non-temporary topic to subscribe to

name

The name used to identify this subscription

### Return Values

TopicSubscriber

Newly created TopicSubscriber, NULL if an error occurs

## US\_createDurableSubscriberWithSelector

### Purpose

Creates a Durable Subscriber to the given Topic with given message selector and local message delivery options.

### Declaration

```
TopicSubscriber US_createDurableSubscriberWithSelector(FioranoSession session, Topic topic, mqstring subscriberID, mqstring messageselector, mqboolean nolocal);
```

### Include

unified\_session.h

### Parameters

session

FioranoSession using which the durable subscriber is to be created

topic

The non-temporary topic to subscribe to

subscriberID

The name used to identify this subscription

messageselector

Only messages with properties matching the message selector expression are delivered

nolocal

If set, inhibits the delivery of messages published on the same connection

### Return Values

TopicSubscriber

Newly created TopicSubscriber, NULL if an error occurs

## US\_createBytesMessage

### Purpose

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

### Declaration

```
Message US_createBytesMessage(FioranoSession session);
```

### Include

```
unified_session.h
```

### Parameters

session

FioranoSession using which the bytes message is to be created

### Return Values

Message

Newly created BytesMessage, NULL if an error occurs

## US\_createMapMessage

### Purpose

Creates a map message. A MapMessage object is used to send a self-defining set of name-value pairs, where names are of type mqstring and values can be of different datatypes.

### Declaration

```
Message US_createMapMessage(FioranoSession session);
```

### Include

```
unified_session.h
```

### Parameters

session

FioranoSession using which the map message is to be created

### Return Values

Message

Newly created MapMessage, NULL if an error occurs

## US\_createStreamMessage

### Purpose

Creates a StreamMessage. A StreamMessage is used to send a self-defining stream of different datatypes.

### Declaration

```
Message US_createStreamMessage(FioranoSession session);
```

### Include

```
unified_session.h
```

### Parameters

session

FioranoSession using which the stream message is to be created

### Return Values

Message

Newly created StreamMessage, NULL if an error occurs

## US\_createTextMessage

### Purpose

Creates a text message. A TextMessage is used to send a message containing an mqstring.

### Declaration

```
Message US_createTextMessage(FioranoSession session);
```

### Include

```
unified_session.h
```

### Parameters

session

FioranoSession using which the text message is to be created

### Return Values

Message

Newly created TextMessage, NULL if an error occurs

## US\_createTextMessageWithText

### Purpose

Creates an initialized text message. A TextMessage is used to send a message containing an mqstring.

### Declaration

```
Message US_createTextMessageWithText(FioranoSession session, mqstring text);
```

### Include

unified\_session.h

### Parameters

session

FioranoSession using which the text message is to be created

text

The string used to initialize this message

### Return Values

Message

Newly created TextMessage, NULL if an error occurs

## US\_commit

### Purpose

Commits all messages done in this transaction and releases any locks held at the time of call of this API.

### Declaration

```
mqboolean US_commit(FioranoSession session);
```

### Include

unified\_session.h

### Parameters

session

FioranoSession to be committed



**Return Values**

TRUE if successful, FALSE if an error occurs

## US\_recover

**Purpose**

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

**Declaration**

```
mqboolean US_rollback(FioranoSession session);
```

**Include**

```
unified_session.h
```

**Parameters**

session

FioranoSession that is to be recovered

**Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## US\_rollback

**Purpose**

Rollbacks any messages done in this transaction and releases any locks held at the time of call of this API.

**Declaration**

```
mqboolean US_recover(FioranoSession session);
```

**Include**

```
unified_session.h
```

**Parameters**

session

FioranoSession to be rollbacked

## Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## US\_close

### Purpose

Closes the FioranoSession and resources allocated on behalf of the unified Session

### Declaration

```
mqboolean US_close(FioranoSession session)
```

### Include

unified\_session.h

### Parameters

session

FioranoSession to be closed

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## US\_free

### Purpose

Frees all the resources allocated on behalf of the FioranoSession.

### Declaration

```
mqboolean US_free(FioranoSession session)
```

### Include

unified\_session.h

### Parameters

session

FioranoSession to be freed

## US\_unsubscribe

### Purpose

Unsubscribes a durable subscription that has been created by a client

This method deletes the state being maintained on behalf of the subscriber by the FioranoMQ server.

It is erroneous for a client to delete a durable subscription while there is an active TopicSubscriber for the subscription, while a consumed message is part of a pending transaction, or has not been acknowledged in the session.

### Declaration

```
mqboolean US_unsubscribe(FioranoSession session, mqstring name)
```

### Include

```
unified_session.h
```

### Parameters

session

FioranoSession using which the subscription is to be unsubscribed

name

The name used to identify this subscription

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## US\_createTopic

### Purpose

Creates a topic identity given a topic name.

### Declaration

```
Topic US_createTopic(FioranoSession session, mqstring topicName)
```

### Include

```
unified_session.h
```

### Parameters

session

FioranoSession using which the topic is to be created

Topicname

Name of the Topic for which reference is to be returned.

### **Return Values**

Topic

A Topic with the given name; NULL if an error occurs

## **US\_createTemporaryTopic**

### **Purpose**

Creates a temporary topic. Lifetime of the topic is that of the FioranoConnection, unless deleted earlier.

### **Declaration**

```
TemporaryTopic US_createTemporaryTopic(FioranoSession session)
```

### **Include**

unified\_session.h

### **Parameters**

session

FioranoSession using which the temporary topic is to be created

### **Return Values**

TemporaryTopic

A temporary topic identity; NULL if an error occurs

## **US\_createQueue**

### **Purpose**

Creates a queue identity given a Queue name.

### **Declaration**

```
Queue US_createQueue(FioranoSession session, mqstring queueName)
```

### **Include**

unified\_session.h

### Parameters

session

FioranoSession using which the queue is to be created

queueName

Name of this queue

### Return Values

Queue

A Queue with the given name

## US\_createTemporaryQueue

### Purpose

Creates a temporary queue. Lifetime of the temporary queue is that of the FioranoConnection, unless deleted earlier.

### Declaration

```
TemporaryQueue US_createTemporaryQueue(FioranoSession session)
```

### Include

unified\_session.h

### Parameters

session

FioranoSession using which the temporary queue is to be created

### Return Values

TemporaryQueue

A temporary queue identity

## US\_createQueueSender

### Purpose

Creates a QueueSender to send messages to the specified destination (queue).

### Declaration

```
QueueSender US_createQueueSender(FioranoSession session, Destination destination)
```

### Include

```
unified_session.h
```

### Parameters

Session

FioranoSession using which the sender is to be created

destination

The destination (queue) to be accessed

### Return Values

QueueSender

Newly created QueueSender, NULL if an error occurs

## US\_createPublisher

### Purpose

Creates a Publisher for the specified destination (topic).

A client uses a FioranoMessageProducerto publish messages on a topic. Each time a client creates a FioranoMessageProduceron a topic, it defines a new sequence of messages that have no ordering relationship with the messages it has previously sent.

### Declaration

```
FioranoMessageProducerUS_createTopicPublisher(FioranoSession session, Destination destination)
```

### Include

```
unified_session.h
```

### Parameters

session

FioranoSession using which the publisher is to be created

destination

The destination (topic) to publish to

### **Return Values**

TopicPublisher

Newly created TopicPublisher, NULL if an error occurs

## **FMP\_send**

### **Purpose**

sends a message to the destination on which the producer is created. Uses the default delivery mode, timeToLive and priority of the destination.

### **Declaration**

```
mqboolean FMP_send(FioranoMessageProducer producer, Message msg)
```

### **Include**

```
fiorano_message_producer.h
```

### **Parameters**

producer

FioranoMessageProducer that is to be used to publish the message

msg

The message to be published

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **FMP\_sendWithParams**

### **Purpose**

sends a message to the destination on which the producer is created. Uses the delivery mode, timeToLive and priority given as arguments.

### **Declaration**

```
mqboolean FMP_sendWithParams(FioranoMessageProducer producer, Message msg, mqint  
deliveryMode, mqint priority, mqlong timeToLive)
```

### **Include**

fiorano\_message\_producer.h

### **Parameters**

producer

FioranoMessageProducer that is to be used to publish the message

msg

The message to be published

deliveryMode

The delivery mode to use

priority

Priority for this message

timeToLive

Lifetime (in milliseconds) of the message

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **FMP\_sendWithDestination**

### **Purpose**

sends a message to the destination specified. Uses the default delivery mode, timeToLive and priority of the destination.

### **Declaration**

```
mqboolean FMP_sendWithDestination(FioranoMessageProducer producer, Destination
destination, Message msg)
```

### **Include**

fiorano\_message\_producer.h

### **Parameters**

producer

FioranoMessageProducer that is to be used to publish the message

destination



Destination on which message is to be published

msg

The message to be published

### **Return Values**

mqboolean

TRUE if successful, FALSE if an error occurs

## **FMP\_sendWithDestinationParams**

### **Purpose**

sends a message to the destination specified. Uses the delivery mode, timeToLive and priority given as arguments.

### **Declaration**

```
mqboolean FMP_sendWithDestinationParams(FioranoMessageProducer producer, Destination destination, Message msg, mqint deliveryMode, mqint priority, mqlong timeToLive)
```

### **Include**

```
fiorano_message_producer.h
```

### **Parameters**

producer

FioranoMessageProducer that is to be used to publish the message

destination

Destination on which message is to be published

msg

The message to be published

deliveryMode

The delivery mode to use

priority

Priority for this message

timeToLive

Lifetime (in milliseconds) of the message

## Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## FMP\_close

### Purpose

Closes the FioranoMessageProducer and the resources allocated on behalf of the FioranoMessageProducer

### Declaration

```
mqboolean FMP_close(FioranoMessageProducer producer)
```

### Include

```
fiorano_message_producer.h
```

### Parameters

producer

FioranoMessageProducer that is to be closed

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## FMP\_free

### Purpose

Frees the resources allocated on behalf of the FioranoMessageProducer.

### Declaration

```
void FMP_free(FioranoMessageProducer producer)
```

### Include

```
fiorano_message_producer.h
```

### Parameters

producer

FioranoMessageProducer that is to be freed

## FMP\_getDeliveryMode

### Purpose

Gets the default delivery mode of the producer.

### Declaration

```
mqint FMP_getDeliveryMode(FioranoMessageProducer producer)
```

### Include

```
fiorano_message_producer.h
```

### Parameters

producer

FioranoMessageProducer whose delivery mode is desired

### Return Values

mqint

The message delivery mode for this message producer

## FMP\_getDisableMessageID

### Purpose

Gets an indication of whether message IDs are disabled.

### Declaration

```
mqboolean FMP_getDisableMessageID(FioranoMessageProducer producer)
```

### Include

```
fiorano_message_producer.h
```

### Parameters

producer

FioranoMessageProducer for which the messageID disable indication has to be checked

### Return Values

mqboolean

An indication of whether message IDs are disabled

## FMP\_getDisableMessageTimestamp

### Purpose

Gets an indication of whether message timestamps are disabled.

### Declaration

```
mqboolean FMP_getDisableMessageTimestamp(FioranoMessageProducer producer)
```

### Include

```
fiorano_message_producer.h
```

### Parameters

producer

FioranoMessageProducer for which the message timestamp indication has to be checked

### Return Values

mqboolean

An indication of whether message IDs are disabled

## FMP\_getPriority

### Purpose

Gets the default priority of the producer.

### Declaration

```
mqint FMP_getPriority(FioranoMessageProducer producer)
```

### Include

```
fiorano_message_producer.h
```

### Parameters

producer

FioranoMessageProducer whose priority is desired

### Return Values

mqint

The message priority for this message producer

## FMP\_getTimeToLive

### Purpose

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

### Declaration

```
mqlong FMP_getTimeToLive(FioranoMessageProducer producer)
```

### Include

```
fiorano_message_producer.h
```

### Parameters

producer

FioranoMessageProducer whose message time to live is desired

### Return Values

mqlong

The message time to live in milliseconds; zero is unlimited

## FMP\_setDeliveryMode

### Purpose

Sets the default delivery mode of the producer.

### Declaration

```
mqboolean FMP_setDeliveryMode(FioranoMessageProducer producer, mqint deliveryMode)
```

### Include

```
fiorano_message_producer.h
```

### Parameters

producer

FioranoMessageProducer for whose messages the delivery mode is to be set.

deliveryMode

The message delivery mode for this message producer; legal values are NON\_PERSISTENT and PERSISTENT.

For more information, read the CRTL Constants section

## Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## FMP\_setDisableMessageID

### Purpose

Sets whether message IDs are disabled for this Topic Publisher.

### Declaration

```
mqboolean FMP_setDisableMessageID(FioranoMessageProducer producer, mqboolean isDisableMessageID)
```

### Include

```
fiorano_message_producer.h
```

### Parameters

producer

FioranoMessageProducer for which the messageID is to be disabled

isDisableMessageID

Indicates if message IDs are disabled

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## FMP\_setDisableMessageTimestamp

### Purpose

Sets whether message timestamps are disabled.

### Declaration

```
mqboolean FMP_setDisableMessageTimeStam(FioranoMessageProducer producer, mqboolean isDisableMsgTimeStam)
```

### Include

```
fiorano_message_producer.h
```

## Parameters

producer

FioranoMessageProducer for which the message timestamp is to be disabled

isDisableMsgTimeStamp

Indicates if message timestamps are disabled

## Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## FMP\_setPriority

### Purpose

Sets the default priority of the producer.

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

### Declaration

```
mqboolean FMP_setPriority(FioranoMessageProducer producer, mqint value)
```

### Include

```
fiorano_message_producer.h
```

### Parameters

producer

FioranoMessageProducer for which the default priority is to be set

value

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

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## FMP\_setTimeToLive

### Purpose

Sets the default length of time in milliseconds, from its dispatch time, for which a produced message should be retained by the message system.

By default, Time to live is set to zero.

### Declaration

```
mqboolean FMP_setTimeToLive(FioranoMessageProducer producer, mqlong value)
```

### Include

```
fiorano_message_producer.h
```

### Parameters

producer

FioranoMessageProducer for which the time to live is to be set  
value

The message time to live in milliseconds; zero is unlimited

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## FMC\_close

### Purpose

Closes the FioranoMessageConsumer and the resources allocated on behalf of the FioranoMessageProducer

### Declaration

```
mqboolean FMC_close(FioranoMessageConsumer consumer)
```

### Include

```
fiorano_message_consumer.h
```

### Parameters

consumer

FioranoMessageConsumer that is to be closed



## Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## FMC\_free

### Purpose

Frees the resources allocated on behalf of the FioranoMessageConsumer.

### Declaration

```
void FMC_free(FioranoMessageConsumer consumer)
```

### Include

```
fiorano_message_consumer.h
```

### Parameters

consumer

FioranoMessageConsumer that is to be freed

## FMC\_start

### Purpose

starts the FioranoMessageConsumer after which it starts receiving messages.

### Declaration

```
mqboolean FMC_start(FioranoMessageConsumer consumer)
```

### Include

```
fiorano_message_consumer.h
```

### Parameters

consumer

FioranoMessageConsumer that is to be started

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## FMC\_stop

### Purpose

Stops the FioranoMessageConsumer after which it cannot receive messages.

### Declaration

```
mqboolean FMC_stop(FioranoMessageConsumer consumer)
```

### Include

```
fiorano_message_consumer.h
```

### Parameters

consumer

FioranoMessageConsumer that is to be stopped

### Return Values

mqboolean

TRUE if successful, FALSE if an error occurs

## Chapter 5: Using the C Runtime Library

---

In this chapter, samples have been used to explain the use of CRTL. In addition to the basic send and receive functionality of the C Runtime Libraries, these samples have been designed to demonstrate features such as

- asynchronous receive
- transacted session

These samples also demonstrate the use of ExceptionListener on a connection and handling error conditions while programming with CRTL. While using these samples, note how the memory allocated for different objects is freed. Similar format can be followed in the user applications as well.

The source code of the sample is as follows.

```
#include <stdio.h>
#include <jms.h>

//Message Listener callback
void OnMessage(Message msg);
//ExceptionListener callback
void OnException(char* exception,void* param);

InitialContextic=NULL;
QueueConnectionFactoryqcf=NULL;
Queue queue= NULL;
QueueConnectionqc=NULL;
QueueSessionsendqs=NULL,

rcvqs= NULL;
QueueSenderqsender=NULL;
QueueReceiver qreceiver = NULL;
TextMessagetmsg= NULL;

/**
 * Free up the memory allocated to the objects and also
 * close the connections, sessions etc.
 */

void cleanup()
{

/**
 * Call the corresponding _free functions for freeing
```

```
* any object not the free() function

*/
IC_free(ic);
Q_free(queue);
QCF_free(qcf);
QSNDR_close(qsender);
QRCVR_close(qreceiver);
QS_close(sendqs);
QS_close(recvqs);
QC_close(qc);
QSNDR_free(qsender);
QRCVR_free(qreceiver);
QS_free(sendqs);
QS_free(recvqs);
QC_free(qc);
}

/**

* The function which is set as the message listener

*/
void OnMessage(Message msg)
{

static mqint msgcount = 0;
++msgcount;
printf("Received Message %d : %s\n", msgcount,
FTMSG_getText(msg));
//Commit after receiving every message from the server
QS_commit(recvqs);

MSG_free(msg);
}

/**

* The function which is set as the exception listener

*/
void OnException(char* exception,void* param)
{

mqstring code = (mqstring)param;
//Print the exception trace and the param and
//cleanup all objects.

printf("Received Message %s : %s\n", param, exception);
cleanup();
exit(-1);

}
```

```
/**
 * After a call to any API, a check should be done for any error
 * occurred in that call. In this example, if any error occurs
 * anywhere in the application, the program will exit
 */

void main()

{
mqint count = 0;
//Create InitialContext

if((ic = newInitialContextDefaultParams()) == NULL)

{
MqPrintException();
MqExceptionClear();
cleanup();
exit(-1);

}

printf("Created the Initial Context\n");
//Lookup for the primaryQCF object on the server
if((qcf = IC_Lookup(ic,"primaryQCF")) == NULL)
{

MqPrintException();
MqExceptionClear();
cleanup();
exit(-1);

}

printf("Looked up primaryQCF\n");

//Lookup for the primaryQueue object on the server
if((queue = IC_Lookup(ic,"primaryQueue")) == NULL)
{

MqPrintException();
MqExceptionClear();
cleanup();
exit(-1);

}

printf("Looked up primaryQueue\n");
//Create a Queue Connection with default parameters
if((qc = QCF_createQueueConnectionDefParams(qcf)) == NULL)
{
```

```
MqPrintException();
MqExceptionClear();
cleanup();
exit(-1);

}
printf("Created the QueueConnection\n");
//Set the ExceptionListener on the connection]
if(QC_setExceptionListener(qc,OnException,"JMSEException
occurred.") == FALSE)

{
MqPrintException();
MqExceptionClear();
cleanup();
exit(-1);

}
//Create a transacted, Auto acknowledgement Queue Session
if((sendqs = QC_createQueueSession(qc, TRUE,

AUTO_ACKNOWLEDGE))
== NULL)

{
MqPrintException();
MqExceptionClear();
cleanup();
exit(-1);

}

//Create a transacted, Auto acknowledgement Queue Session
if((recvqs = QC_createQueueSession(qc, TRUE,
AUTO_ACKNOWLEDGE))
== NULL)

{
MqPrintException();
MqExceptionClear();
cleanup();
exit(-1);

}

//Create a Sender to send messages on the primaryQueue
if((qsender = QS_createSender(sendqs, queue)) == NULL)
{

MqPrintException();
MqExceptionClear();
cleanup();
exit(-1);
```

```
}
printf("Created a sender to send messages on primary-
Queue\n");

//Create a Receiver to receive messages on the primaryQueue
if((qreceiver = QS_createReceiver(recvqs, queue)) == NULL)
{

MqPrintException();
MqExceptionClear();
cleanup();
exit(-1);

}

//Starting the Connection, essential to start receiving

messages
if(!QC_start(qc))
{

MqPrintException();
MqExceptionClear();
cleanup();
exit(-1);

}
//Create a TextMessage without any text in it
if((tmsg = QS_createTextMessage(sendqs)) == NULL)
{

MqPrintException();
MqExceptionClear();
cleanup();
exit(-1);

}

//Send 10 Text Messages using the qsender
for(count = 0; count < 10; count ++)
{

if(!FTMSG_setText(tmsg, "FioranoMQ"))

{
MqPrintException();
MqExceptionClear();
cleanup();

exit(-1);
}
}
```

```
// Sending a message using the Sender
if(!QSNDR_send(qsender, tmsg))
{

MqPrintException();
MqExceptionClear();
cleanup();
exit(-1);

}
}

MSG_free(tmsg);

// Commit the send session so that all the messages are sent
to
// the FMQ Server
if(!QS_commit(sendqs))
{

MqPrintException();
MqExceptionClear();
cleanup();
exit(-1);

}
printf("***** Commit the sender session *****\n");
//Waiting for 5 messages on the Queue (Synchronous receive)
for(count = 0; count < 5; count++)
{

if((tmsg = QRCVR_receive(qreceiver)) == NULL)

{
MqPrintException();
MqExceptionClear();
cleanup();
exit(-1);

}

printf("Received message : %s\n", FTMSG_getText(tmsg));
MSG_free(tmsg);

}

//Rollback the Receiver Session, so that all the messages /
/received are rolled-back
if(!QS_rollback(recvqs))
{
```



```
MqPrintException();
MqExceptionClear();
cleanup();
exit(-1);

}
printf("*****Rollback the receiver session *****\n");

//Setting a message listener, for asynchronous receive
if(!QRCVR_setMessageListener(qreceiver, OnMessage))
{

MqPrintException();
MqExceptionClear();
cleanup();

exit(-1);
}

//Wait for the messages to be received by the message lis-
tener
Sleep(10000);
//Cleanup all the resources allocated in the sample
cleanup();

}
```

The following steps explain the sequence of calls taking place in the sample:

- Declare callback functions for both the MessageListener and the Exception-Listener.
- An InitialContext is created to lookup the Queue and QueueConnectionFactory objects from the FioranoMQ server.
- The QueueConnectionFactory is used to create a QueueConnection.
- The ExceptionListener callback declared earlier is set as the exception listener for the QueueConnection with a constant mqstring as the parameter.
- The QueueConnection is then used to create two sessions, one for sending the messages and the other for receiving the messages. The sessions are created in transacted mode, to use the commit/rollback feature.
- A QueueSender is created on the queue "primaryqueue" using one of the sessions.
- A QueueReceiver is created on the queue "primaryqueue" using the other session.
- The QueueSender is used to send 10 messages. After sending the messages, the sender session is committed, so as to send all the messages to the FioranoMQ server.
- Then the QueueReceiver is used to receive 5 messages, using the QRCVR\_receive() API, which is the synchronous way of receiving messages.
- The messages received by the receiver are rolled-back, when the QS\_rollback() API is invoked for the receiver session.

- The message listener callback declared earlier is set on the receiver, which receives 10 messages, since the first 5 messages were rolled-back.
- The receiver session commits after receiving every message.
- The user should destroy all the resources allocated in the application, used the corresponding APIs defined by CRTL. So this is accomplished by Cleanup() API, which closes the connection, sessions, senders and receivers, and destroy the various objects.

For more sample applications check the \c\samples directory of the FioranoMQ installation.

## Organization of Samples Provided

The CRTL samples provided in the FioranoMQ installation are organized into the following directories that can be found in the /crtl/samples directory of the installation.

### ptp

- basic Demonstrates the basic JMS Send/Receive functionality
- browser Demonstrates the use of QueueBrowser
- transaction Demonstrates the commit/rollback functionality in QueueSession
- http Demonstrates the use of http
- https Demonstrates the use of https
- msgsel Demonstrates the use of message properties and message selectors
- reqrep Demonstrates the use of the JMS request/reply API (the request() and replyTo() APIs)
- ssl Demonstrates the use of ssl

### pubsub

- basic Demonstrates the basic JMS publish/subscribe functionality
- reqrep Demonstrates the Request/Reply mechanism
- msgsel Demonstrates the use of message properties and message selectors
- transaction Demonstrates the commit/rollback functionality in TopicSession
- dursub Demonstrates the basic Durable Subscriber functionality
- http Demonstrates the use of http
- https Demonstrates the use of https
- ssl Demonstrates the use of ssl

## Compiling and Running the Samples

The following sections contain the procedures for compiling and running the samples. The samples on Windows and Solaris.

### On Windows

#### Compiling the samples

The samples provided with CRTL can be compiled using the script `cclientbuild.bat`, provided along with the C-runtime package. The following variables in the script file should be set to the appropriate paths:

- **VC\_DIR** The root directory of the Microsoft Visual C++ installation
- **INCLUDE\_PATH** The directory where all the header files of CRTL exist
- **CRTL\_LIB** The directory where the `fmq-crtl.lib` file and other required libraries exist.

Libraries required for compiling the CRTL samples (apart from the default VC libs):

- `Advapi32.lib`
- `ws2_32.lib`
- `fmq-crtl.lib`
- `crtl_https.lib` (only for SSL and HTTP/HTTPS samples)
- `pthreadVC.lib`
- `libeay32.lib` (only for SSL samples)
- `ssleay32.lib` (only for SSL samples)
- `http.lib` (only for HTTP/HTTPS samples)
- `gnu_regex.lib` (only for HTTP/HTTPS samples)
- `zlib.lib` (only for HTTP/HTTPS samples)

Use the Multi-threaded option for Code generation in the project.

The following example illustrates how to compile a sample using `cclientbuild.bat` file:

```
cclientbuild Sample.c
```

After compilation a Win32 executable is formed that can be executed as explained in the following section.

For more details on compiling and running the samples, please read the `readme.txt` files provided in `c\samples` folder of FioranoMQ installation.

## Running the Samples

After compiling the samples as explained above, they can be executed by using the corresponding executable file generated.

Sample [`<arg1>` `<arg2>` ... ]

The following DLLs need to in the system PATH for running the compiled WIN32 executable:

- pthreadVC.dll
- libeay32.dll (only for SSL samples)
- ssleay32.dll (only for SSL samples)
- gnu\_regex.dll (only for HTTP/HTTPS samples)
- zlib.dll (only for HTTP/HTTPS samples)

## On Solaris

### Compiling the samples

The samples provided with CRTL can be compiled using the script `cclientbuild.sh`, provided along with the C-runtime package. This script file uses the GCC compiler and assumes that the same is available on the concerned Solaris machine.

The following variables in the script file should be set to the appropriate paths:

- `CRTL_HOME` The base directory of the CRTL package in the FioranoMQ installation

Libraries required for compiling the CRTL samples:

- `libsocket.so`
- `libpthread.so`
- `libnsl.so`
- `libcrtl.so`
- `libcrtl_https.so` (only for SSL and HTTP/HTTPS samples)
- `libhttp.so` (only for HTTP/HTTPS samples)
- `libcrypto.a` (only for HTTP/HTTPS samples)
- `libssl.a` (only for SSL and HTTP/HTTPS samples)

The following example illustrates how to compile a sample using `cclientbuild.sh` file:

```
cclientbuild.sh Sample.c
```

After compilation, a bin file (`Sample.bin` in the above example) is formed that can be executed as explained in the following section.

### Running the Samples

The bin file formed after compilation can be executed from the console as follows.

```
Sample.bin [<arg1> <arg2> ... ]
```

# Chapter 6: Native C-Runtime Examples

---

This guide explains you about the sample programs used for C-RTL for PubSub and PTP operations.

## PTP

- Admin
- Browser
- Message Selectors
- RequestReply
- TimedRequestReply
- SendReceive
- Transactions
- Http
- Https
- SSL

## PubSub

- Admin
- Durable Subscribers
- Message Selectors
- Pubsub
- RequestReply
- TimedRequestReply
- Transactions
- Http
- Https
- SSL

These samples are available in %FMQ\_DIR%\clients\c\native\samples directory.

## PTP Samples

### Admin

This directory contains one sample programs which illustrate basic JMS Administration API functionality using the FioranoMQ C Runtime Library.

- **AdminTest.c** - Uses C Native RTL Administration API to create and delete Queues, QueueConnectionFactory and retrieves information of users connected from the server.

To run these samples using FioranoMQ, do the following:

1. Compile the source files. For convenience, compiled version of the sources are included in this directory. The %FMQ\_DIR%\clients\c\native\scripts directory contains a script called clientbuild.bat which compiles the C program.
2. Run the AdminTest by executing the AdminTest.exe executable file.

**Note:** To run any of the C samples, please ensure that environment variable FMQ\_DIR points to Fiorano's installation directory.

(Default: c:\progra~1\Fiorano\FioranoMQ9\fmq)

### Browser

This directory contains two sample programs which illustrate basic JMS Browser functionality using the FioranoMQ C Runtime Library.

- **Sender.c** - Reads strings from standard input and sends them on the queue "primaryqueue".
- **Browser.c** - Implements a browser, which is used to browse the messages on the queue "primaryqueue", and prints out the received messages.

To run these samples using FioranoMQ, do the following:

1. Compile each of the source files. For convenience, compiled version of the sources is included in this directory. The %FMQ\_DIR%\c\native\scripts directory contains a script called client-build.bat which compiles the C program.
2. Run the Sender by executing the Sender.exe executable file. Send some messages before running the browser application
3. Run the asynchronous receiver by executing the Browser.exe file.

**Note:** To run any of the C samples, ensure that environment variable FMQ\_DIR points to Fiorano's installation directory (this defaults to \Fiorano\FioranoMQ\fmq)

## msgsel

This directory contains two sample programs which illustrate the use of message selectors using the FioranoMQ C Runtime Library.

- **SelectorSend.c** - Selector sends messages with the string property "name" and an int property "value", set differently for 3 consecutive messages.
- **SelectorRecv.c** - Implements an asynchronous listener, which listens on the queue "primaryqueue" for the messages which match the criteria specified in the message selector, and prints out the received messages.

To run these samples using FioranoMQ, perform the following steps:

1. Compile each of the source files. For convenience, compiled version of the sources is included in this directory. The %FMQ\_DIR%\c\native\scripts directory contains a script called client-build.bat which compiles the C program.
2. Run the Sender by executing the SelectorSend.exe executable file.
3. Run the asynchronous receiver by executing the selectorRecv.exe file.

**Note:** To run any of the C samples, ensure that environment variable FMQ\_DIR points to Fiorano's installation directory (this defaults to \Fiorano\FioranoMQ\fmq)

## reqrep

### basic

This directory contains two sample programs which illustrate the request/reply abstraction supplied by the JMS API using a C requestor application and replying application

- **Requestor.c** - Reads strings from standard input and use it to send a request message on the queue "primaryqueue".
- **Replier.c** - Implements an asynchronous listener, which listens on the queue "primaryqueue", and replies to each request

To run these samples using FioranoMQ, perform the following steps:

1. Compile each of the source files. For convenience, compiled version of the sources is included in this directory. The %FMQ\_DIR%\c\native\scripts directory contains a script called cclient-build.bat which compiles the C program.
2. Run the Replier by executing the Replier.exe executable file.
3. Run the requestor by executing the Requestor.exe file.

**Note:** To run any of the C samples, ensure that environment variable FMQ\_DIR points to Fiorano installation directory (this defaults to \Fiorano\FioranoMQ\fmq).



## timeout

This directory contains two sample programs which illustrate the timed request/ reply abstraction provided by FioranoMQ.

- **TimedRequestor.c** - Reads strings from standard input and use it to send a request on the queue "primaryqueue". It waits for the reply, for a specific time interval.
- **TimedReplier.c** - Implements an asynchronous listener, which listens on the queue "primaryqueue", and replies to each request.

To run these samples using FioranoMQ, perform the following steps:

1. Compile each of the source files. For convenience, compiled version of the %FMQ\_DIR%\c\native\scripts directory contains a script called cclient-build.bat which compiles the C program.
2. Run the replier by executing the TimedReplier.exe executable file.
3. Run the timed requestor by executing the TimedRequestor.exe file.

**Note:** To run any of the C samples, ensure that environment variable FMQ\_DIR points to Fiorano's installation directory (this defaults to \Fiorano\FioranoMQ\fmq).

## basic

This directory contains two sample programs which illustrate basic JMS Send/ Receive functionality using the FioranoMQ C Runtime Library.

- **Sender.c** - Reads strings from standard input and sends them on the queue "primaryqueue".
- **Receiver.c** - Implements an asynchronous listener, which listens on the queue "primaryqueue", and prints out the received messages.

To run these samples using FioranoMQ, perform the following steps:

1. Compile each of the source files. For convenience, compiled version of the sources is included in this directory. The %FMQ\_DIR%\c\native\scripts directory contains a script called cclient-build.bat which compiles the C program.
2. Run the Sender by executing the *sender.exe* executable file.
3. Run the asynchronous receiver by executing the Receiver.exe file.

**Note:** To run any of the C samples, ensure that environment variable FMQ\_DIR points to Fiorano installation directory (this defaults to \Fiorano\FioranoMQ\fmq).

## Transactions

This directory contains a sample programs which illustrate JMS Transaction functionality using the FioranoMQ C Runtime Library.

- **Transactions.c** - Implements the sender and receiver, and uses the commit/rollback functionality to demonstrate JMS Transactions

To run this sample using FioranoMQ, do the following:

1. Compile the source files. For convenience, compiled version of the sources is included in this directory. The %FMQ\_DIR%\c\native\scripts directory contains a script called cclient-build.bat which compiles the C program.
2. Run the sample by executing the *transactions.exe* executable file. For proper results from the sample, ensure that there are no messages in the primary-Queue.

**Note:** To run any of the C samples, ensure that environment variable FMQ\_DIR points to Fiorano's installation directory (this defaults to \Fiorano\FioranoMQ\fmq)

## Http

This directory contains two sample programs which illustrate the use of HTTP protocol for basic JMS ptp functionality using the FioranoMQ C Runtime Library.

- **HttpReceiver.c** - Receives messages asynchronously on primaryQueue. This program implements an asynchronous listener to listen for messages published on the queue primaryQueue.
- **HttpSender.c** - Implements a client application publishing user specified data on primaryQueue. This program reads strings from standard input and publishes them on the Queue primaryQueue.

To run this sample using FioranoMQ, do the following:

1. Compile the source file. For convenience, compiled version of the sources is included in this directory. The %FMQ\_DIR%\c\native\scripts directory contains a script called cclientbuild.bat which compiles the C program.
2. Run the HttpReceiver by executing the *httpReceiver.exe* executable file.
3. Run the HttpSender by executing the *httpSender.exe* executable file.

**Note:** To run any of the C samples, ensure that environment variable FMQ\_DIR points to Fiorano installation directory (this defaults to \Fiorano\FioranoMQ\fmq). Runtime dependency on Win Platform is on following files [c\native\lib]:

- gnu\_regex.dll
- libeay32.dll
- ssleay32.dll
- zlib.dll

These are required to be present in path [either in same directory or in WINDOWS\System32].

## Https

This directory contains two sample programs which illustrate the use of HTTPS protocol for basic JMS ptp functionality using the FioranoMQ C Runtime Library.

- **HttpsReceiver.c**- Receives messages asynchronously on primaryQueue. This program implements an asynchronous listener to listen for messages published on the queue primaryQueue.
- **HttpsSender.c**- Implements a client application publishing user specified data on primaryQueue. This program reads strings from standard input and publishes them on the Queue primaryQueue.

To run this sample using FioranoMQ, do the following:

1. Compile the source file. For convenience, compiled version of the sources is included in this directory. The %FMQ\_DIR%\c\native\scripts directory contains a script called cclientbuild.bat which compiles the C program.
2. Run the HttpsReceiver by executing the *httpsReceiver.exe* executable file.
3. Run the HttpsSender by executing the *httpsSender.exe* executable file.

**Note:** To run any of the C samples, ensure that environment variable FMQ\_DIR points to Fiorano installation directory (this defaults to \Fiorano\FioranoMQ\fmq). Runtime dependency on Win Platform is on following files [c\native\lib]:

- gnu\_regex.dll
- libeay32.dll
- ssleay32.dll
- zlib.dll

These are required to be present in path [either in same directory or in WINDOWS\System32].

## SSL

This directory contains two sample programs which illustrate the basic JMS pub-sub and ptp functionality over Secure Socket Layer using the FioranoMQ C Runtime Library.

- **SSLReceiver.c** - Receives messages asynchronously on primaryQueue. This program implements an asynchronous listener to listen for messages published on the queue primaryQueue.
- **SSLSender.c** - Implements a client application publishing user specified data on primaryQueue. This program reads strings from standard input and publishes them on the Queue primaryQueue. To run this sample using FioranoMQ, do the following:

To run this sample using FioranoMQ, do the following:

1. Compile the source file. For convenience, compiled version of the sources, are included in this directory. The %FMQ\_DIR%\c\native\scripts directory contains a script called cclientbuild.bat which compiles the C program.
2. Run the SSLReceiver by executing the *sslReceiver.exe* executable file.
3. Run the SSLSender by executing the *sslSender.exe* executable file.

**Note:** To run any of the C samples, ensure that environment variable FMQ\_DIR points to Fiorano installation directory (this defaults to \Fiorano\FioranoMQ\fmq). Runtime dependency on Win Platform is on following files [c\native\lib]:

- gnu\_regex.dll
- libeay32.dll
- ssleay32.dll
- zlib.dll

These are required to be present in path [either in same directory or in WINDOWS\System32].

## PubSub Samples

### Admin

This directory contains one sample programs which illustrate basic JMS Administration API functionality using the FioranoMQ C Runtime Library.

- **AdminTest.c**- Uses C Native RTL Administration API to create and delete Topics, TopicConnectionFactory and retrieves information from the server.

To run these samples using FioranoMQ, do the following:

1. Compile the source files. For convenience, compiled version of the sources, are included in this directory. The %FMQ\_DIR%\clients\c\native\scripts directory contains a script called cclientbuild.bat which compiles the C program.
2. Run the AdminTest by executing the *adminTest.exe* executable file.

**Note:** To run any of the C samples, please ensure that environment variable FMQ\_DIR points to Fiorano's installation directory.

(Defaults: c:\progra~1\Fiorano\FioranoMQ9\fmq)

### dursub

This directory contains two sample programs which illustrate basic JMS Durable-Subscriber functionality using the FioranoMQ C Runtime Library.

- **Publisher\_d.c** - Reads strings from standard input and publishes PERSISTENT messages on the topic "primarytopic".
- **DurableSubscriber.c** - Implements a durable subscriber using the client ID "DS\_Client\_1", and durable subscriber name "Sample\_Durable\_Subscriber", listening on the topic "primarytopic".

To run these samples using FioranoMQ, perform the following steps:

1. Compile each of the source files. For convenience, compiled version of the sources is included in this directory. The %FMQ\_DIR%\c\native\scripts directory contains a script called cclient-build.bat which compiles the C program.

2. Start the DurableSubscriber program first, so that the subscriber can register with the FioranoMQ Server.
3. Next, start the Publisher\_d program. When the program comes up, type in a few strings, pressing the Enter key after each string. The string is published and is received by the Durable Subscriber started in step (a) above.
4. Now, shut down the Durable Subscriber, but keep typing in messages into the Publisher program. These messages are automatically stored by the FioranoMQ Server, since a Durable Subscriber was previously registered on the topic to which the messages are being published.
5. After a while, restart the DurableSubscriber program. On restart, you would find that all messages that were published during the time that the durable subscriber was down are now made available to the subscriber.
6. Repeat steps 4 and 5 over. Each time, you would find that all messages published during the time that the Subscriber is down are immediately made available to the Subscriber when it restarts.

**Note:** To run any of the C samples, ensure that environment variable FMQ\_DIR points to Fiorano installation directory (this defaults to \Fiorano\FioranoMQ\fmq).

## msgsel

This directory contains two sample programs which illustrate the use of message selectors using the FioranoMQ C Runtime Library.

- **SelectorSend.c** - Selector sends messages with the string property "name" and an int property "value", set differently for 3 consecutive messages.
- **SelectorRecv.c** - Implements an asynchronous listener, which listens on the topic "primarytopic" for the messages which match the criteria specified in the message selector, and prints out the received messages.

To run these samples using FioranoMQ, perform the following steps:

1. Compile each of the source files. For convenience, compiled version of the sources is included in this directory. The %FMQ\_DIR%\c\native\scripts directory contains a script called cclient-build.bat which compiles the C program.
2. Run the Sender by executing the *selectorSend.exe* executable file.
3. Run the asynchronous receiver by executing the *selectorRecv.exe* file.

**Note:** To run any of the C samples, ensure that environment variable FMQ\_DIR points to Fiorano's installation directory (default is \Fiorano\FioranoMQ).

## basic

This directory contains two sample programs which illustrate basic JMS Publisher/ Subscriber functionality using the FioranoMQ C Runtime Library.

- **Publisher.c** - Reads strings from standard input and sends them on the topic "primarytopic".

- **Subscriber.c** - Implements an asynchronous listener, which listens on the topic "primarytopic", and prints out the received messages.

To run these samples using FioranoMQ, perform the following steps:

1. Compile each of the source files. For convenience, compiled version of the sources is included in this directory. The %FMQ\_DIR%\c\native\scripts directory contains a script called cclient-build.bat which compiles the C program.
2. Run the Publisher by executing the *publisher.exe* executable file.
3. Run the asynchronous subscriber by executing the *subscriber.exe* file.

**Note:** To run any of the C samples, ensure that environment variable FMQ\_DIR points to Fiorano installation directory (default is \Fiorano\FioranoMQ).

## reqrep

### basic

This directory contains two sample programs which illustrate the request/reply abstraction supplied by the JMS API using a C requestor application and replying application

- **Requestor.c** - Reads strings from standard input and use it to send a request message on the topic "primarytopic".
- **Replier.c** - Implements an asynchronous listener, which listens on the topic "primarytopic", and replies to each request

To run these samples using FioranoMQ, perform the following steps:

1. Compile each of the source files. For convenience, compiled version of the sources is included in this directory. The %FMQ\_DIR%\c\native\scripts directory contains a script called cclient-build.bat which compiles the C program.
2. Run the Replier by executing the *replier.exe* executable file.
3. Run the requestor by executing the *requestor.exe* file.

**Note:** To run any of the C samples, ensure that environment variable FMQ\_DIR points to Fiorano installation directory (this default to \Fiorano\FioranoMQ).

### timeout

This directory contains two sample programs which illustrate the request/reply abstraction supplied by the JMS API using a C requestor application and replying application

- **TimedRequestor.c** - Reads strings from standard input and use it to send a request message on the topic "primarytopic".
- **TimedReplier.c** - Implements an asynchronous listener, which listens on the topic "primarytopic", and replies to each request

To run these samples using FioranoMQ, perform the following steps:

1. Compile each of the source files. For convenience, compiled version of the sources is included in this directory. The %FMQ\_DIR%\c\native\scripts directory contains a script called cclient-build.bat which compiles the C program.
2. Run the Replier by executing the *timedReplier.exe* executable file.
3. Run the requestor by executing the *timedRequestor.exe* file.

**Note:** To run any of the C samples, ensure that environment variable FMQ\_DIR points to Fiorano installation directory (this defaults to \Fiorano\FioranoMQ\fmq).

## Transaction

This directory contains a sample programs which illustrate JMS Transaction functionality using the FioranoMQ C Runtime Library.

- **Transactions.c** - Implements the sender and receiver, and uses the commit/rollback functionality to demonstrate JMS Transactions

To run this sample using FioranoMQ, do the following:

1. Compile the source file. For convenience, compiled version of the sources is included in this directory. The %FMQ\_DIR%\c\native\scripts directory contains a script called cclient-build.bat which compiles the C program.
2. Run the Replier by executing the Transactions.exe executable file. For proper results from the sample, ensure that there are no messages in the primary-Topic.

**Note:** To run any of the C samples, ensure that environment variable FMQ\_DIR points to Fiorano installation directory (this defaults to \Fiorano\FioranoMQ\fmq).

## Http

This directory contains two sample programs which illustrate the use of HTTP protocol for basic JMS pubsub functionality using the FioranoMQ C Runtime Library.

- **HttpSubscriber.c** - Receives messages asynchronously published on "primary-Topic". This program implements an asynchronous listener to listen for messages published on "primaryTopic".
- **HttpPublisher.c** - Implements a client application publishing user specified data on "primaryTopic". This program reads strings from standard input and publishes them on "primaryTopic".

To run this sample using FioranoMQ, do the following:

1. Compile the source file. For convenience, compiled version of the sources is included in this directory. The %FMQ\_DIR%\c\native\scripts directory contains a script called cclientbuild.bat which compiles the C program.
2. Run the HttpSubscriber by executing the *httpSubscriber.exe* executable file.
3. Run the HttpPublisher by executing the *httpPublisher.exe* executable file.

**Note:** To run any of the C samples, ensure that environment variable FMQ\_DIR points to Fiorano installation directory (this defaults to \Fiorano\FioranoMQ\fmq). Runtime dependency on Win Platform is on following files [c\native\lib]:

- gnu\_regex.dll
- libeay32.dll
- ssleay32.dll
- zlib.dll

These are required to be present in path [either in same directory or in WINDOWS\System32].

## Https

This directory contains two sample programs which illustrate the use of HTTPS protocol for basic JMS pubsub functionality using the FioranoMQ C Runtime Library.

- **HttpsSubscriber.c** - Receives messages asynchronously published on "primaryTopic". This program implements an asynchronous listener to listen for messages published on "primaryTopic".
- **HttpsPublisher.c** - Implements a client application publishing user specified data on "primaryTopic". This program reads strings from standard input and publishes them on "primaryTopic".

To run this sample using FioranoMQ, do the following:

1. Compile the source file. For convenience, compiled version of the sources is included in this directory. The %FMQ\_DIR%\c\native\scripts directory contains a script called cclientbuild.bat which compiles the C program.
2. Run the HttpsSubscriber by executing the *httpsSubscriber.exe* executable file.



3. Run the `HttpsPublisher` by executing the `httpsPublisher.exe` executable file.

**Note:** To run any of the C samples, ensure that environment variable `FMQ_DIR` points to Fiorano installation directory (this defaults to `\Fiorano\FioranoMQ\fmq`). Runtime dependency on Win Platform is on following files [`c\native\lib`]:

- `gnu_regex.dll`
- `libeay32.dll`
- `ssleay32.dll`
- `zlib.dll`

These are required to be present in path [either in same directory or in `WINDOWS\System32`].

## ssl

This directory contains two sample programs which illustrate the basic JMS pub-sub and ptp functionality over Secure Socket Layer using the FioranoMQ C Runtime Library.

- **SSLSubscriber.c** - Receives messages asynchronously published on "primary-Topic". This program implements an asynchronous listener to listen for messages published on "primaryTopic".
- **SSLPublisher.c** - Implements a client application publishing user specified data on "primaryTopic". This program reads strings from standard input and publishes them on "primaryTopic".

To run this sample using FioranoMQ, do the following:

1. Compile the source file. For convenience, compiled version of the sources, are included in this directory. The `%FMQ_DIR%\c\native\scripts` directory contains a script called `cclientbuild.bat` which compiles the C program.
2. Run the `SSLSubscriber` by executing the `sslSubscriber.exe` executable file.
3. Run the `SSLPublisher` by executing the `sslPublisher.exe` executable file.

**Note:** To run any of the C samples, ensure that environment variable `FMQ_DIR` points to Fiorano installation directory (this defaults to `\Fiorano\FioranoMQ\fmq`). Runtime dependency on Win Platform is on following files [`c\native\lib`]:

- `gnu_regex.dll`
- `libeay32.dll`
- `ssleay32.dll`
- `zlib.dll`

These are required to be present in path [either in same directory or in `WINDOWS\System32`].