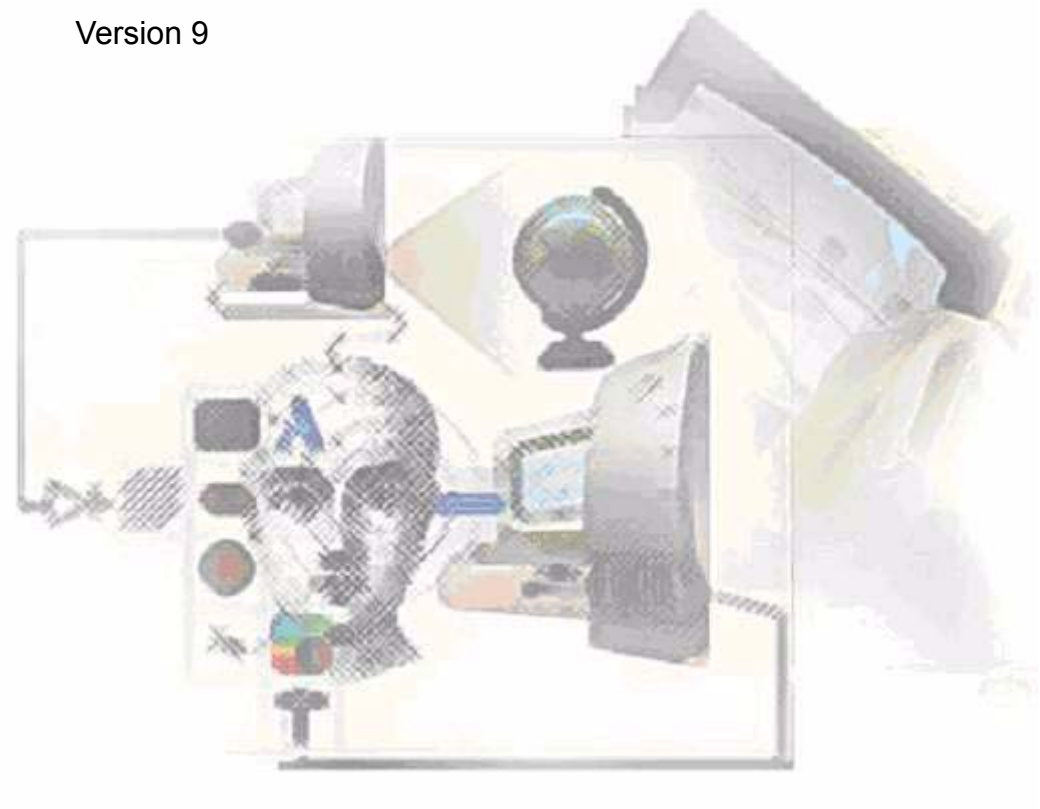
Fiorano Software, Inc.

FioranoMQ 9 – Messaging Server

C(Cross-compiled) RTL Guide

Version 9



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

Contents

1

Preface

Audience

This guide is designed to assist developers in using FMQ C JNI/Cross-compiled runtime library for enabling communication between C and Java applications. The developer must have a fair knowledge of the C programming language.

Overview of the Guide

This guide contains the following contents:

- An introduction to FioranoMQ JNI/Cross-compiled CRuntime library.
- Detailed description on how to use the APIs.
- Details on compiling and running C RTL sample applications.

Introduction to FMQ C(JNI) RTL

FioranoMQ has a full featured (including XA, Administration APIs, Client Side persistence) C client runtime library, enabling C programs to use FMQ's JMS messaging facilities through a thin set of wrapper classes. This RTL can be run with or without a JRE. When run without a JRE, the RTL uses a dynamically loaded library (DLL) which contains the cross-compiled version of Java platform classes and Fiorano's Java RTL.

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

Typographical Conventions

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

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

TABLE 1 Typographical Conventions

Related Documentation

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

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

Document Name	Description
Native C# Runtime Library Guide	Contains a description of the FioranoMQ C# Runtime APIs.
JavaDocs	Contains "javadoc" style documentation on all public Fiorano classes.

TABLE 2 Related Documentation

2

Administration APIs

This chapter explains how the RTL can be initialized and also describes the Administration Application Programming Interfaces in CJNI RTL. The various classes included in this package are:

- MQAdminConnectionFactory
- MQAdminConnection
- MQAdminService
- MQDispatcherService
- MQLogService
- MQMonitoringService
- MQNamingService
- MQSnooperService
- MQTraceService
- MQRealmService
- JMSMetaData
- QueueMetaData
- TopicMetaData
- Topic
- MessageProducer
- Queue
- ConnectionFactoryMetaData
- AdminConnectionFactoryMetaData
- QueueConnectionFactoryMetaData
- TopicConnectionFactoryMetaData
- UnifiedXAConnectionFactoryMetaData

- XAQueueConnectionFactoryMetaData
- XATopicConnectionFactoryMetaData
- ServerMetaData

Initialization and Finalizing APIs

To access any of the functions exposed by the C Cross-compiled Runtime library, you need to execute the following functions.

initFioranoCRT(int mode)

This function initializes the RTL and should be called before accessing any other function in this RTL. The value of the parameter "mode" determines whether the client will run using the JRE or using a dynamically loaded library (DLL). The DLL contains cross-compiled version of Java platform classes and Fiorano's Java RTL.

This function accepts any one of following values for the mode parameter.

- `USE_JRE` To initialize the RTL to use JRE
- `USE_NATIVE_CODE` To initialize the RTL to use native DLL

finalizeFioranoCRT()

This function should be called by the client app just before exiting the RTL.

Administration APIs

MQAdminConnectionFactory

Connection factory is a JMS administered Object which is typically looked using JNDI.

mqacf_createMQAdminConnection

Description

Gets a TopicConnection object from the factory.

Function Signature

```
jint mqacf_createMQAdminConnection(FHandle mqacf, FHandle mqac,
const char username, const char password);
```

Parameters

`mqacf` - The MQ admin connection factory to operate on.

`username` - The caller's username.

`password` - The caller's password.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

MQAdminConnection

This class provides methods for getting instances of different services required by a FioranoMQ administrator to perform various admin activities like creating admin objects, users and groups; setting permissions for users and monitoring snooping and Dispatcher related activities.

mqac_close

Description

Closes the Admin connection with the FMQ Server and cleans up all related resources.

Function Signature

```
jint mqac_close(FHandle mqac);
```

Parameters

`mqac` - The MQ admin connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error

mqac_getConnectionProperties

Description

Gets the Server properties.

Function Signature

```
jint mqac_getConnectionProperties(FHandle mqac, FHandle hashtable);
```

Parameters

`mqac` - The MQ admin connection to operate on.

Returns

`MQ_SUCCESS` if successful, MQ_FAILURE if error.

mqac_getConnectUrl

Description

Function Signature

This method returns the url to which FMQ client is currently connected.

Function Signature

```
jint mqac_getConnectUrl(FHandle mqac, const char * url);
```

Parameters

`mqac` - The MQ admin connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqac_getCurrentConnectURL

Description

This method is called by AdminGUI to display the current URL to which the connection is made.

Function Signature

```
jint mqac_getCurrentConnectURL(FHandle mqac, const char *url);
```

Parameters

`mqac` - The MQ admin connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqac_getMQAdminService

Description

Returns a handle to the MQAdminService that is used to perform various admin activities like creating admin objects, getting active clientIDs, getting registered subscriptions, getting the number of messages in queue/topic etc.

Function Signature

```
jint mqac_getMQAdminService(FHandle mqac, FHandle mqAdminService);
```

Parameters

`mqac` - The MQ admin connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqac_getMQDispatcherService

Description

Returns a handle to the MQDispatcherService that is used to manage the FMQ Dispatcher related activities in the FioranoMQ server.

Function Signature

```
jint mqac_getMQDispatcherService(FHandle mqac, FHandle mqDispatch-
erService);
```

Parameters

`mqac` - The MQ admin connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqac_getMQLogService

Description

Returns a handle to the MQLogService that is used to control/monitor the logging operations performed by the LogManager in the FioranoMQ server.

Function Signature

```
jint mqac_getMQLogService(FHandle mqac, FHandle mqLogService);
```

Parameters

`mqac` - The MQ admin connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqac_getMQMonitoringService

Description

Returns a handle to the MQMonitoringService that is used to manage different events and monitoring related activities in the FioranoMQ server.

Function Signature

```
jint mqac_getMQMonitoringService(FHandle mqac, FHandle mqMonitor-
ingService);
```

Parameters

`mqac` - The MQ admin connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqac_getMQNamingService

Description

Returns a handle to the MQNamingService that is used to bind and lookup admin objects to and from the JNDI store being used in the FioranoMQ server.

Function Signature

```
jint mqac_getMQNamingService(FHandle mqac, FHandle mqNamingSer-
vice);
```

Parameters

`mqac` - The MQ admin connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqac_getMQRealmService

Description

Returns a handle to the MQRealmService that is used to manage users and groups and their corresponding permissions in the FioranoMQ server.

Function Signature

```
jint mqac_getMQRealmService(FHandle mqac, FHandle mqRealmService);
```

Parameters

`mqac` - The MQ admin connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqac_getMQSnooperService

Description

Returns a handle to the MQSnooperService that is used to manage the FioranoMQ Snooper related activities in the FioranoMQ server.

Function Signature

```
jint mqac_getMQSnooperService(FHandle mqac, FHandle mqSnooperSer-
vice);
```

Parameters

`mqac` - The MQ admin connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqac_getMQTraceService

Description

Returns a handle to the MQTraceService class that is used to set and get trace levels for different components of the FioranoMQ server.

Function Signature

```
jint mqac_getMQTraceService(FHandle mqac, FHandle mqTraceService);
```

Parameters

`mqac` - The MQ admin connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqac_getParentConnectionFactory

Description

Returns the parent connection factory for this admin connection.

Function Signature

```
jint mqac_getParentConnectionFactory(FHandle mqac, FHandle
parentCF);
```

Parameters

`mqac` - The MQ admin connection to operate on.

`parentCF` - Contains the parent connection factory.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqac_getServerProperties

Description

Gets the server properties for this MQAdminConnection.

Function Signature

```
jint mqac_getServerProperties(FHandle mqac, FHandle hashtable);
```

Parameters

`mqac` - The MQ admin connection to operate on.

`hashtable` - Contains the server properties.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqac_revalidate

Description

For Admin reconnection purpose.

Function Signature

```
jint mqac_revalidate(FHandle mqac);
```

Parameters

`mqac` - The MQ admin connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqac_setExceptionListener

Description

Set's ExceptionListener callback.

Function Signature

```
jint mqac_setExceptionListener(FHandle mqac, voidlistener) ;
```

Parameters

`mqac` - The MQ admin connection to operate on.

`listener` - The callback function.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqac_startReconnectThread

Description

Starts the thread to reconnect.

Function Signature

```
jint mqac_startReconnectThread(FHandle mqac);
```

Parameters

`mqac` - The MQ admin connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqac_stopReconnectThread

Description

It stops the reconnect thread.

Function Signature

```
jint mqac_stopReconnectThread(FHandle mqac);
```

Parameters

`mqac` - The MQ admin connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

MQAdminService

This class provides methods for performing various administrative tasks on the FioranoMQ server. These tasks include creation and deletion of administered objects, getting the active client ids and users, getting registered topic subscriptions, getting message count on a queue or topic etc.

mqadminservice_createAdminConnectionFactory

Description

Creates a new Admin Connection Factory on the FMQ server.

Function Signature

```
jint mqadminservice_createAdminConnectionFactory(FHandle mqas,
FHandle metaData);
```

Parameters

`mqas` - The MQ admin service to operate on.

`metaData` - AdminConnectionFactoryMetaData object using which a AdminConnectionFactory is created.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_createConnectionFactory

Description

Creates a new unified Connection Factory.

Function Signature

```
jint mqadminservice_createConnectionFactory(FHandle mqas, FHandle
metaData);
```

Parameters

`mqas` - The MQ admin service to operate on.

`metaData` - UnifiedConnectionFactoryMetaData object using which a CommonConnectionFactory is created.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_createQueue

Description

Creates a queue on the Server using the argument metadata.

Function Signature

```
jint mqadminservice_createQueue(FHandle mqas, FHandle metaData);
```

Parameters

`mqas` - The MQ admin service to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_createQueueConnectionFactory

Description

Creates a new Queue Connection Factory on the FMQ server.

Function Signature

```
jint mqadminservice_createQueueConnectionFactory(FHandle mqas,
FHandle metaData);
```

Parameters

`mqas` - The MQ admin service to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_createTopic

Description

Creates a Topic on the Server using the argument metadata.

Function Signature

```
jint mqadminservice_createTopic(FHandle mqas, FHandle metaData);
```

Parameters

`mqas` - The MQ admin service to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_createTopicConnectionFactory

Description

Creates a new Topic Connection Factory on the FMQ server.

Function Signature

```
jint mqadminservice_createTopicConnectionFactory(FHandle mqas,
FHandle metaData);
```

Parameters

`mqas` - the MQ admin service to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_createXAConnectionFactory

Description

Creates a new unified XA Connection Factory.

Function Signature

```
jint mqadminservice_createXAConnectionFactory(FHandle mqas, FHandle
metaData);
```

Parameters

`mqas` - The MQ admin service to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_createXAQueueConnectionFactory

Description

Creates a new XAQueue Connection Factory.

Function Signature

```
jint mqadminservice_createXAQueueConnectionFactory(FHandle mqas,
FHandle metaData);
```

Parameters

`mqas` - The MQ admin service to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_createXATopicConnectionFactory

Description

Creates a new XATopic Connection Factory.

Function Signature

```
jint mqadminservice_createXATopicConnectionFactory(FHandle mqas,
FHandle metaData);
```

Parameters

`mqas` -The MQ admin service to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_currentUsers

Description

Gets an enumeration of names of all the active users on the FMQ server.

Function Signature

```
jint mqadminservice_currentUsers(FHandle mqas, FHandle enumera-
tion);
```

Parameters

`mqas` - The MQ admin service to operate on.

`enumeration` - An enumeration of all logged-in or active users.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_deleteAdminConnectionFactory

Description

Delete an admin connection factory from the Server.

Function Signature

```
jint mqadminservice_deleteAdminConnectionFactory(FHandle mqas,
const charFunction Signature name);
```

Parameters

`mqas` - The MQ admin service to operate on.

`name` - The name of the admin connection factory to be deleted.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_deleteConnectionFactory

Description

Delete a queue connection factory from the Server.

Function Signature

```
jint mqadminservice_deleteConnectionFactory(FHandle mqas, const
charFunction Signature name);
```

Parameters

`mqas` - The MQ admin service to operate on.

`name` - The name of the queue connection factory to be deleted.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_deleteMessagesOnQueue

Description

This AdminAPI is used for deleting the messages From Queue based on startIndex and EndIndex.

Function Signature

```
jint mqadminservice_deleteMessagesOnQueue(FHandle mqas, const char-
Function Signature queueName, jlong startIndex, jlong endIndex, jint
priority, jintFunction Signature noOfMessages);
```

Parameters

`mqas` - The MQ admin service to operate on.

`noOfMessages` - The no of messages to be deleted.

`queueName` - The name of the queue to be deleted.

`startIndex` - The start index of the range.

`endIndex` - The the end index of the range.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_deleteQueue

Description

Delete a Queue from the Server.

Function Signature

```
jint mqadminservice_deleteQueue(FHandle mqas, const charFunction
Signature name);
```

Parameters

`mqas` - The MQ admin service to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_deleteQueueConnectionFactory

Description

Delete a QueueConnectionFactory from the Server.

Function Signature

```
jint mqadminservice_deleteQueueConnectionFactory(FHandle mqas,
const charFunction Signature name);
```

Parameters

`mqas` - The MQ admin service to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_deleteTopic

Description

Delete a Topic from the Server.

Function Signature

```
jint mqadminservice_deleteTopic(FHandle mqas, const charFunction
Signature name);
```

Parameters

`mqas` - The MQ admin service to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE MQ_FAILURE if error.

mqadminservice_deleteTopicConnectionFactory

Description

Delete a TopicConnectionFactory from the Server.

Function Signature

```
jint mqadminservice_deleteTopicConnectionFactory(FHandle mqas,
const charFunction Signature name);
```

Parameters

`mqas` - The MQ admin service to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_deleteXAConnectionFactory

Description

Delete a QueueConnectionFactory from the Server.

Function Signature

```
jint mqadminservice_deleteXAConnectionFactory(FHandle mqas, const
charFunction Signature name);
```

Parameters

`mqas` - The MQ admin service to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_deleteXAQueueConnectionFactory

Description

Delete a XAQueueConnectionFactory from the Server.

Function Signature

```
jint mqadminservice_deleteXAQueueConnectionFactory(FHandle mqas,
const charFunction Signature name);
```

Parameters

`mqas` - The MQ admin service to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_deleteXATopicConnectionFactory

Description

Delete a XATopicConnectionFactory from the Server.

Function Signature

```
jint mqadminservice_deleteXATopicConnectionFactory(FHandle mqas,
const charFunction Signature name);
```

Parameters

`mqas` - The MQ admin service to operate on.

`name` - Name of topic connection factory to be deleted.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_elements

Description

Gets an Enumeration of administered Objects of specified type.

Function Signature

```
jint mqadminservice_elements(FHandle mqas, jbyte type, FHandleFunc-
tion Signature enumeration);
```

Parameters

`mqas` - The MQ admin service to operate on.

`type` - Type of admin object passed as byte.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_getAllTransactions

Description

Gets an enumeration of the Xids of all transactions of a particular status.

Function Signature

```
jint mqadminservice_getAllTransactions(FHandle mqas, jbyte status,
FHandleFunction Signature enumeration);
```

Parameters

`mqas` - The MQ admin service to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_getClientIDs

Description

Gets an Enumeration of clientIDs of all connections using which a durable subscription has been registered on the FMQ server.

Function Signature

```
jint mqadminservice_getClientIDs(FHandle mqas, FHandle enumeration);
```

Parameters

`mqas` - The MQ admin service to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_getDurableSubscribersForTopic

Description

Gets an Enumeration of IDs of all durable subscribers for the topic

Function Signature

```
jint mqadminservice_getDurableSubscribersForTopic(FHandle mqas,
const charFunction Signature topicName, FHandleFunction Signature
enumeration);
```

Parameters

`mqas` - The MQ admin service to operate on.

`topicName` - The topic name for which to get the number of durable subscribers.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_getNamingManagerImpl

Description

This method is used by Admin GUI, to obtain the implementation class of Naming Services at the Server.

Function Signature

```
jint mqadminservice_getNamingManagerImpl(FHandle mqas, const char-
Function Signature *namingManager);
```

Parameters

`mqas` - The MQ admin service to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_getNoOfActiveClientConnections

Description

Gets the number of Active Client Connections on the FioranoMQ server.

Function Signature

```
jint mqadminservice_getNoOfActiveClientConnections(FHandle mqas,
jintFunction Signature noOfActiveClientConnections);
```

Parameters

`mqas` - The MQ admin service to operate on.

`noOfActiveClientConnections` - The number of active client connections.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_getNumberOfDeliverableMessages

Description

Gets the number of messages available in the given queue.

Function Signature

```
jint mqadminservice_getNumberOfDeliverableMessages(FHandle mqas,
const charFunction Signature queueName, jlongFunction Signature
num);
```

Parameters

`mqas` - The MQ admin service to operate on.

Parameters

`num` - The number of messages undelivered to the given Queue.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_getNumberOfDeliverableMessages_1

Description

Gets the number of messages deliverable (includes both undelivered and redeliverable) to the given client-subscriberID.

Function Signature

```
jint mqadminservice_getNumberOfDeliverableMessages_1(FHandle mqas,
const charFunction Signature clientID, const charFunction Signature
subscriberID, jlongFunction Signature num);
```

Parameters

`mqas` - The MQ admin service to operate on.

`clientID` - The client's identification string.

`subscriberID` - The subscriber's identification string.

`num` - The number of messages undelivered to the given client-subscriber ID.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_getNumberOfUndeletedMessages

Description

Gets the number of undeleted messages in a given queue.

Function Signature

```
jint mqadminservice_getNumberOfUndeletedMessages(FHandle mqas,
const charFunction Signature queueName, jintFunction Signature num);
```

Parameters

`mqas` - The MQ admin service to operate on.

`queueName` - Name of the queue as a string.

`num` - The number of undeleted messages in the queue.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_getPTPClientIDs

Description

Gets an Enumeration of all the active PTP clientIDs.

Function Signature

```
jint mqadminservice_getPTPClientIDs(FHandle mqas, FHandleFunction
Signature enumeration);
```

Parameters

`mqas` - The MQ admin service to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_getPubSubClientIDs

Description

Gets an Enumeration of all the active Pubsub clientIDs

Function Signature

```
jint mqadminservice_getPubSubClientIDs(FHandle mqas, FHandleFunc-
tion Signature enumeration);
```

Parameters

`mqas` - The MQ admin service to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_getSubscriberIDs

Description

Gets an Enumeration of all the subscriberIDs registered using a particular clientID.

Function Signature

```
jint mqadminservice_getSubscriberIDs(FHandle mqas, const charFunc-
tion Signature clientID, FHandleFunction Signature enumeration);
```

Parameters

`mqas` - The MQ admin service to operate on.

`clientID` - The client ID of the connection passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_getSubscriptionTopicName

Description

Gets the name of a topic on which a particular subscription is registered.

Function Signature

```
jint mqadminservice_getSubscriptionTopicName(FHandle mqas, const
charFunction Signature clientID, const charFunction Signature sub-
scriberID, const charFunction Signature *name);
```

Parameters

`mqas` - The MQ admin service to operate on.

`clientID` - clientID of the connection.

`subscriberID` - the subscriber ID of the subscriber.

`name` - Contains the topic name on which the durable subscriber identified by the pair [clientID, subscriberID] was created after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_getTransactionStatus

Description

Gets the status of a particular transaction (XA Resource) from the FMQ server.

Function Signature

```
jint mqadminservice_getTransactionStatus(FHandle mqas, FHandle xid,
jbyte trStatus);
```

Parameters

`mqas` - The MQ admin service to operate on.

`xid` - The xid of the transaction.

`trStatus` - Contains the status of the particular transaction after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_purgeQueueMessages

Description

Purge all the messages of the queue if there are no active consumers on that queue.

Function Signature

```
jint mqadminservice_purgeQueueMessages(FHandle mqas, const char-
Function Signature queueName);
```

Parameters

`mqas` - The MQ admin service to operate on.

`queueName` - Name of the queue whose messages are to be purged.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_purgeSubscriptionMessages

Description

Deletes all the pending messages for the durable subscription identified by the pair [clientID,subscriberID].

Function Signature

```
jint mqadminservice_purgeSubscriptionMessages(FHandle mqas, const
charFunction SignatureclientID, const charFunction Signature sub-
scriberID);
```

Parameters

`mqas` - The MQ admin service to operate on.

`clientID` - ClientID of the connection.

`subscriberID` - SubscriberID of the durable subscriber.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_showStatusOfAllQueues

Description

Test Method that can be used to check the status of all the queues registered in qgms.

Function Signature

```
jint mqadminservice_showStatusOfAllQueues(FHandle mqas);
```

Parameters

`mqas` - The MQ admin service to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_shutDownActiveHAServer

Description

Shutdown the Active HAServer.

Function Signature

```
jint mqadminservice_shutDownActiveHAServer(FHandle mqas);
```

Parameters

`mqas` - The MQ admin service to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_shutDownPassiveHAServer

Description

Shutdown the Passive HAServer.

Function Signature

```
jint mqadminservice_shutDownPassiveHAServer(FHandle mqas);
```

Parameters

`mqas` - The MQ admin service to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_shutdownServer

Description

Shutdown the FioranoMQ Server.

Function Signature

```
jint mqadminservice_shutdownServer(FHandle mqas);
```

Parameters

`mqas` - The MQ admin service to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqadminservice_unsubscribe

Description

Unsubscribes the durable subscription identified by the pair [clientID, subscriberID].

Function Signature

```
jint mqadminservice_unsubscribe(FHandle mqas, const charFunction
Signature clientID, const charFunction Signature subscriberID);
```

Parameters

`mqas` - The MQ admin service to operate on.

`clientID` - clientID of the connection.

`subscriberID` - subscriberID of the durable subscriber.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

MQDispatcherService

This class defines the Dispatcher APIs that can be used to modify or maintain the EMS servers in a dispatcher cluster.

mqdispatcherservice_addServerToCluster

Description

Add the server specified by the ServerConfigData to the cluster managed by this Dispatcher

Function Signature

```
jint mqdispatcherservice_addServerToCluster(FHandle mqds, FHandle
server, const char loginName, const char passwd, const char admin-
ConnectionFactory);
```

Parameters

`mqds` - The MQDispatcherService to operate on.

`server` - A ServerMetaData object.

`login` - The username.

`passwd` - The password.

`adminConnectionFactory` - ACF with which Dispatcher Connects with the member server.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqdispatcherservice_getActiveServersInCluster

Description

Gets an enumeration of all the Active Servers in the Server cluster.

Function Signature

```
jint mqdispatcherservice_getActiveServersInCluster(FHandle mqds,
FHandle result);
```

Parameters

`mqds` - the MQDispatcherService to operate on.

`result` - Contains an Enumeration object containing ServerMetaData of all servers that are Active in the cluster.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqdispatcherservice_getPreferredServer

Description

Gets the Server config data of the preferred server of the dispatcher cluster.

Function Signature

```
jint mqdispatcherservice_getPreferredServer(FHandle mqds, FHandle
result);
```

Parameters

`mqds` - The MQDispatcherService to operate on.

`result` - Contains the ServerMetaData of the Preferred Server in the Cluster

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqdispatcherservice_getServersInCluster

Description

Gets an Enumeration of ServerMetaData representing all the Servers in the cluster catered by this Dispatcher.

Function Signature

```
jint mqdispatcherservice_getServersInCluster(FHandle mqds, FHandle
result);
```

Parameters

`mqds` - The MQDispatcherService to operate on.

`result` - Contains an Enumeratino object of all the servers in this cluster.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqdispatcherservice_removeServerFromCluster

Description

Removes the Server specified by the ServerConfigData from the server cluster managed by this dispatcher.

Function Signature

```
jint mqdispatcherservice_removeServerFromCluster(FHandle mqds,
FHandle server);
```

Parameters

`mqds` - The MQDispatcherService to operate on.

`server` - A URL object representing the URL of the server.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqdispatcherservice_saveDispatcherConfigurations

Description

Saves the configurations of the Dispatcher in the XML config File.

Function Signature

```
jint mqdispatcherservice_saveDispatcherConfigurations(FHandle
mqds);
```

Parameters

`mqds` - The MQDispatcherService to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqdispatcherservice_setAsPreferredServer

Description

The admin can set the preferred server for the cluster This is the server to which the clients will be connected while creating a Durable Subscription.

Function Signature

```
jint mqdispatcherservice_setAsPreferredServer(FHandle mqds, FHandle server);
```

Parameters

`mqds` - The MQDispatcherService to operate on.

`server` - A URL object representing the URL of the server.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqdispatcherservice_setMaximumClientConnections

Description

The admin can set or change the maximum client connections for a member server.

Function Signature

```
jint mqdispatcherservice_setMaximumClientConnections(FHandle mqds, FHandle server, jint conns);
```

Parameters

`mqds` - The MQDispatcherService to operate on.

`server` - A URL object representing the URL of the FioranoEMS server.

`conns` - Maximum client connections.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqdispatcherservice_setPreferredServer

Description

The admin can set the preferred server for the cluster This is the server to which the clients will be connected while creating a Durable Subscription.

Function Signature

```
jint mqdispatcherservice_setPreferredServer(FHandle mqds, FHandle
serverMetaData, const char loginName, const char passwd, const char
adminConnectionFactory);
```

Parameters

`mqds` - The MQDispatcherService to operate on.

`serverMetaData` - The ServerMetaData object of the FioranoEMS server.

`loginName` - The username.

`passwd` - The password.

`adminConnectionFactory` - ACF with which Dispatcher Connects with the member server.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

MQLogService

This class provides APIs for viewing the content of FioranoMQ log files.

mqlogservice_getErrorLog

Description

Returns the error log available in specified fileNo.

Function Signature

```
jint mqlogservice_getErrorLog(FHandle mqls, jint fileNo, const char
*result);
```

Parameters

`mqls` - The MQLogService to operate on.

`fileNo` - File number for which to get the error log.

`result` - Contains the out log available in the specified file number.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqlogservice_getNoOfFiles

Description

Gets the number of log files formed at the FMQ server.

Function Signature

```
jint mqlogservice_getNoOfFiles(FHandle mqls, jint result);
```

Parameters

`mqls` - The MQLogService to operate on.

`result` - Contains the number of log files used at the server end.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqlogservice_getOutLog

Description

Returns the Out log available in specified fileNo.

Function Signature

```
jint mqlogservice_getOutLog(FHandle mqls, jint fileNo, const char
*result);
```

Parameters

`mqls` - The MQLogService to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

MQMonitoringService

This class provides APIs for starting and managing system events that monitor server health.

mqmonitoringservice_addMonitor

Description

Adds an event of a particular type for Monitoring purposes.

Function Signature

```
jint mqmonitoringservice_addMonitor(FHandle mqms, jint type, jlong
timeout);
```

Parameters

`mqms` - The MQMonitoringService to operate on.

`type` - Type of system event.

`timeout` - Timeout period of the event.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqmonitoringservice_removeMonitor

Description

Removes a specified system event.

Function Signature

```
jint mqmonitoringservice_removeMonitor(FHandle mqms, jint type);
```

Parameters

`mqms` - The MQMonitoringService to operate on.

`type` - The type of event.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqmonitoringservice_startSystemEvents

Description

Starts off system events for Monitoring the server health.

Function Signature

```
jint mqmonitoringservice_startSystemEvents(FHandle mqms);
```

Parameters

`mqms` - The MQMonitoringService to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqmonitoringservice_stopSystemEvents

Description

Stops all system events that monitor the server health.

Function Signature

```
jint mqmonitoringservice_stopSystemEvents(FHandle mqms);
```

Parameters

`mqms` - The MQMonitoringService to operate on.

Returns

MQ_SUCCESS If successful, MQ_FAILURE if error.

MQNamingService

This class is used to bind, lookup and destory JMS “Administered” objects on the FioranoMQ server

mqnamingservice_bind

Description

Add the Admin Object specified by given AdminObjectName to the list of authorised Administrated Objects that can be used by Client Applications that connect to this Server

Function Signature

```
jint mqnamingservice_bind(FHandle mqns, const char adminObjectName,
FHandle data, jboolean result);
```

Parameters

`mqns` - The MQNamingService to operate on.

`adminObjectName` - Name of the object that is to be bound to the admin store.

`result` - TRUE if object is successfully bound; FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqnamingservice_destroy

Description

Remove the adminObject identified by a given adminObjectName from the list of administered Objects on the server.

Function Signature

```
jint mqnamingservice_destroy(FHandle mqns, const charFunction Signature adminObjectName, jboolean result);
```

Parameters

`mqns` - The MQNamingService to operate on.

`adminObjectName` - The name of the admin object to destroy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqnamingservice_elements

Description

Returns an enumeration of all Adminobjects(either Destination or ConnectionFactory) on the FMQ server.

Function Signature

```
jint mqnamingservice_elements(FHandle mqns, FHandle result);
```

Parameters

`mqns` - The MQNamingService to operate on.

`result` - An Enumeration object containing all admin objects.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqnamingservice_lookup

Description

Looks up an adminobject using the specified name from the FMQ server.

Function Signature

```
jint mqnamingservice_lookup(FHandle mqns, const char adminObject-
Name, FHandle result);
```

Parameters

`mqns` - The MQNamingService to operate on.

`adminObjectName` - The name of the admin object to look up.

`result` - Contains the object that was looked up.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

MQSnooperService

This class provides methods for Admin requests to perform message snooping on destinations. MQSnooperService allows Administrator to selectively view JMS-Messages flowing on any Topic or Queue.

mqsnooperservice_getRegisteredSnooperDestinations

Description

Gets an enumeration of all the registered snoopers.

Function Signature

```
jint mqsnooperservice_getRegisteredSnooperDestinations(FHandle
mqss, FHandle result);
```

Parameters

`mqss` - The MQSnooperService to operate on.

`result` - Contains an Enumeration object of all the registered snoopers.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqsnooperservice_isDestinationRegistered

Description

Checks if the snooper is registered for the specified destination

Function Signature

```
jint mqsnooperservice_isDestinationRegistered(FHandle mqss, const
char destinationName, jboolean result);
```

Parameters

`mqss` - The MQSnooperService to operate on.

`destinationName` - The name of the destination.

`result` - Contains a boolean indicating whether the snooper is registered for the specified destination.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqsnooperservice_isSnooperStarted

Description

Checks whether snooping is started on the registered destinations or not.

Function Signature

```
jint mqsnooperservice_isSnooperStarted(FHandle mqss, jboolean re-
sult);
```

Parameters

`mqss` - The MQSnooperService to operate on.

`result` - Will be TRUE if the snooper service is started.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqsnooperservice_registerMessageSnooperForAll

Description

Registers the snooper for all the destinations on the FMQ server.

Function Signature

```
jint mqsnooperservice_registerMessageSnooperForAll(FHandle mqss);
```

Parameters

`mqss` - The MQSnooperService to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqsnooperservice_registerMessageSnooperForDestination

Description

Registers the snooper for a particular destination.

Function Signature

```
jint mqsnooperservice_registerMessageSnooperForDestination(FHandle
mqss, const char destinationName);
```

Parameters

`mqss` - The MQSnooperService to operate on.

`destinationName` - The destination's name.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqsnooperservice_saveSnooperConfigurations

Description

Saves the latest configurations of the Snooper in the XML config File.

Function Signature

```
jint mqsnooperservice_saveSnooperConfigurations(FHandle mqss);
```

Parameters

`mqss` - The MQSnooperService to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqsnooperservice_startSnooperForAll

Description

Starts off the snoopers on all the destinations which have registered for snooping.

Function Signature

```
jint mqsnooperservice_startSnooperForAll(FHandle mqss);
```

Parameters

`mqss` - The MQSnooperService to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqsnooperservice_stopSnooperForAll

Description

Stops the snoopers on all the destinations which have been started.

Function Signature

```
jint mqsnooperservice_stopSnooperForAll(FHandle mqss);
```

Parameters

`mqss` - The MQSnooperService to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqsnooperservice_unregisterMessageSnooperForAll

Description

Unregisters the snooper for all the destinations which have been registered.

Function Signature

```
jint mqsnooperservice_unregisterMessageSnooperForAll(FHandle mqss);
```

Parameters

`mqss` - The MQSnooperService to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqsnooperservice_unregisterMessageSnooperForDestination

Description

Unregisters the snooper for a particular destination.

Function Signature

```
jint mqsnooperservice_unregisterMessageSnooperForDestination(FHandle mqss, const char destinationName);
```

Parameters

`mqss` - The MQSnooperService to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

MQTraceService

This class provides APIs to manage tracing of various modules of the FMQ server. Using these APIs administrators can change the trace levels of any module/component of the FMQ server.

mqtraceservice_getAllTraceComponents

Description

Gets all the traceable components of the FMQ server.

Function Signature

```
jint mqtraceservice_getAllTraceComponents(FHandle mqts, FHandle result);
```

Parameters

`mqts` - The MQTraceService to operate on.

`result` - Contains a Hashtable object containing all the traceable components with their trace values.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqtraceservice_getLevelForComponent

Description

Gets the trace level for a trace component.

Function Signature

```
jint mqtraceservice_getLevelForComponent(FHandle mqts, const char
component, jint result);
```

Parameters

`mqts` - The MQTraceService to operate on.

`component` - The name of the component.

`result` - Contains the level of the trace component.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqtraceservice_saveTraceConfigurations

Description

Saves the latest trace levels in the trace.cfg file at the server.

Function Signature

```
jint mqtraceservice_saveTraceConfigurations(FHandle mqts);
```

Parameters

`mqts` - The MQTraceService to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqtraceservice_trace

Description

Sets tracing for all components at specified level.

Function Signature

```
jint mqtraceservice_trace(FHandle mqts, jint level);
```

Parameters

`mqts` - The MQTraceService to operate on.

`level` - The level.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqtraceservice_trace_1

Description

Sets tracing for specified component at specified level.

Function Signature

```
jint mqtraceservice_trace_1(FHandle mqts, const char componentName,
jint level);
```

Parameters

`mqts` - The MQTraceService to operate on.

`componentName` - The name of the component.

`level` - The trace level (0-6).

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

MQRealmService

This class provides methods for managing the realm storage through the FioranoMQ server. This involves creation and deletion of users and groups,creating and managing ACLs and ACEs.

mqrealmservice_authunticateUser

Description

Authenticates the user.

Function Signature

```
jint mqrealmservice_authunticateUser(FHandle mqrs, const char user-
name, const char passwd, jboolean result);
```

Parameters

`mqrs` - The MQRealmService to operate on.

`username` - The username.

`passwd` - The password.

`result` - Contains TRUE if the username with the given password exists, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqrealmservice_changePassword

Description

Changes the password of a user.

Function Signature

```
jint mqrealmservice_changePassword(FHandle mqrs, const char user-
name, const char newPasswd, jboolean result);
```

Parameters

`mqrs` - The MQRealmService to operate on.

`username` - The username.

`newPasswd` - The new password.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqrealmservice_createAcl

Description

Creates an ACL and intializes the owner of the ACL.

Function Signature

```
jint mqrealmservice_createAcl(FHandle mqrs, const char name, FHandle
owner, FHandle result);
```

Parameters

`mqrs` - The MQRealmService to operate on.

`name` - The name of the ACL.

`owner` - The Principal owner of the ACL.

`result` - Contains a newly created Acl object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqrealmservice_createAclEntry

Description

Creates an empty Acl Entry object.

Function Signature

```
jint mqrealmservice_createAclEntry(FHandle mqrs, FHandle result);
```

Parameters

`mqrs` - The MQRealmService to operate on.

`result` - Contains an AclEntry

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqrealmservice_createGroup

Description

Creates a Group with the specified name in the database.

Function Signature

```
jint mqrealmservice_createGroup(FHandle mqrs, const char name, FHan-
dle result);
```

Parameters

`mqrs` - The MQRealmService to operate on.

`result` - Contains a newly created empty Group object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqrealmservice_createUser

Description

Creates a new User with the specified Name.

Function Signature

```
jint mqrealmservice_createUser(FHandle mqrs, const char username,
const char passwd, FHandle result);
```

Parameters

`mqrs` - The MQRealmService to operate on.

`username` - The username.

`passwd` - The password.

`result` - Contains a Principal object representing this user.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqrealmservice_deleteGroup

Description

Deletes a group, if it is empty and is a member of no group or ACL.

Function Signature

```
jint mqrealmservice_deleteGroup(FHandle mqrs, const char name,
jboolean result);
```

Parameters

`mqrs` - The MQRealmService to operate on.

`name` - The name of the Group.

`result` - Contains TRUE ,iff the group is successfully deleted.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqrealmservice_deleteUser

Description

Deletes the identified User, if it is supported by the underlying realm implementation.

Function Signature

```
jint mqrealmservice_deleteUser(FHandle mqrs, const char name, jboolean result);
```

Parameters

`mqrs` - The MQRealmService to operate on.

`name` - The name of the user to be deleted.

`result` - Contains TRUE, iff the user is successfully deleted.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqrealmservice_getAcl

Description

Gets the Acl associated the particular context(principal) which is passed as a context name.

Function Signature

```
jint mqrealmservice_getAcl(FHandle mqrs, const char aclName, FHandle result);
```

Parameters

`mqrs` - The MQRealmService to operate on.

`aclName` - The name of the Acl.

`result` - Contains the Acl object associated with the particular context.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqrealmservice_getAclNames

Description

Returns the names of all the ACLs in this particular realm.

Function Signature

```
jint mqrealmservice_getAclNames(FHandle mqrs, FHandle result);
```

Parameters

`mqrs` - The MQRealmService to operate on.

`result` - Contains an Enumeration object of the Acl names.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqrealmservice_getDefaultAclOwner

Description

Gets default ACL Owner.

Function Signature

```
jint mqrealmservice_getDefaultAclOwner(FHandle mqrs, FHandle re-
sult);
```

Parameters

`mqrs` - The MQRealmService to operate on.

`result` - Contains the Principal which, by default, owns the ACLs in the realm.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqrealmservice_getGroup

Description

Gets Group with the specified Name.

Function Signature

```
jint mqrealmservice_getGroup(FHandle mqrs, const char username,
FHandle result);
```

Parameters

`mqrs` - The MQRealmService to operate on.

`username` - The name of the Group that has to be retrieved.

`result` - Contains the Group object representing this name.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqrealmservice_getGroupNames

Description

Returns the names of all the groups in this particular realm.

Function Signature

```
jint mqrealmservice_getGroupNames(FHandle mqrs, FHandle result);
```

Parameters

`mqrs` - The MQRealmService to operate on.

`result` - Contains an Enumeration object of all the group names.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqrealmservice_getPermission

Description

Gets the Permission Object representing the Permission of the specified type.

Function Signature

```
jint mqrealmservice_getPermission(FHandle mqrs, const char type,
FHandle result);
```

Parameters

`mqrs` - The MQRealmService to operate on.

`type` - The type of Permission.

`result` - Contains the Permission object of the specified type.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqrealmservice_getUser

Description

Gets User with the specified Name.

Function Signature

```
jint mqrealmservice_getUser(FHandle mqrs, const char username, FHan-
dle result);
```

Parameters

`mqrs` - The MQRealmService to operate on.

`username` - The username.

`result` - Contains the Principal object representing the user.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqrealmservice_getUserNames

Description

Returns the names of all the users in this particular realm.

Function Signature

```
jint mqrealmservice_getUserNames(FHandle mqrs, FHandle result);
```

Parameters

`mqrs` - The MQRealmService to operate on.

`result` - Contains an Enumeration object of all the usernames.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

mqrealmservice_setACL

Description

Sets the ACL to the required Context(principal).

It also updates existing ACLs for the required Context.

Function Signature

```
jint mqrealmservice_setACL(FHandle mqrs, const char context, FHandle
acl, jboolean result);
```

Parameters

`mqrs` - The MQRealmService to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

JMSMetaData

This class is the base class for all the MetaData classes supported by MQ.

createJMSMetaData

Description

Creates a new JMS meta data object.

Function Signature

```
jint createJMSMetaData(FHandle jmsMetaData);
```

Parameters

`jmsMetaData` - A reference to the newly created object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

jmsmetadata_equals

Description

Compares the given object with this object.

Function Signature

```
jint jmsmetadata_equals(FHandle jmd, FHandle obj, jboolean result);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`obj` - The object to compare.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

jmsmetadata_getDescription

Description

Gets the description of the Administered Object The object implements the read-External method to restore its contents by calling the methods of DataInput for primitive types and readObject for objects, strings and arrays.

Function Signature

```
jint jmsmetadata_getDescription(FHandle jmd, const char *descrip-
tion);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`description` - A description of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

jmsmetadata_getName

Description

Gets the Name of the Administered Object.

Function Signature

```
jint jmsmetadata_getName(FHandle jmd, const char *name);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`name` - The name of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

jmsmetadata_hashCode

Description

Returns a hash code value for the object.

Function Signature

```
jint jmsmetadata_hashCode(FHandle jmd, jint hc);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`hc` - The has code value of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

jmsmetadata_setDescription

Description

Sets the description of the MetaData Object

Function Signature

```
jint jmsmetadata_setDescription(FHandle jmd, const char strDescrip-
tion);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`strDescription` - A description for this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

jmsmetadata_setName

Description

Sets the Name of this MetaData Object

Function Signature

```
jint jmsmetadata_setName(FHandle jmd, const char strName);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`strName` - The name of this meta data object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

jmsmetadata_validateMetaData

Description

Validates if the metaData name is null or NOT.

Function Signature

```
jint jmsmetadata_validateMetaData(FHandle jmd);
```

Parameters

`jmd` - The JMS meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

QueueMetaData

This class represents the MetaData information for a Queue at it is stored in an LDAP directory. Objects of this class are used to create Queues.

createQueueMetaData

Description

Creates a new queue meta data object.

Function Signature

```
jint createQueueMetaData(FHandle qmd);
```

Parameters

`qmd` - A reference to the newly created object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuemetadata_enableCompression

Description

Enablesmessage compression on this queue using the default level and strategy.

Function Signature

```
jint queuemetadata_enableCompression(FHandle qmd);
```

Parameters

`qmd` - The queue meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuemetadata_enableCompression_1

Description

Enablesmessage compression on this queue using the level and strategy passed as parameters This would mean that all messages published on this queue will be compressed using the level and strategy specified in this API.

Function Signature

```
jint queuemetadata_enableCompression_1(FHandle qmd, jint level, jint
strategy);
```

Parameters

`qmd` - The queue meta data object to operate on.

`level` - CompressionLevel (0-9)

`strategy` - CompressionStrategy

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuemetadata_equals

Description

Compares the given object with this object.

Function Signature

```
jint queuemetadata_equals(FHandle qmd, FHandle obj, jboolean re-
sult);
```

Parameters

`qmd` - The queue meta data object to operate on.

`obj` - The object to compare with.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuemetadata_getAlgo

Description

Gets the algorithm used to encrypt data published on this queue.

Function Signature

```
jint queuemetadata_getAlgo(FHandle qmd, const char *algo);
```

Parameters

`qmd` - The queue meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuemetadata_getCompressionLevel

Description

Gets the compression level using which the messages sent on this Queue will be compressed.

Function Signature

```
jint queuemetadata_getCompressionLevel(FHandle qmd, jint level);
```

Parameters

`qmd` - The queue meta data object to operate on.

Function Signature

`@level` - Level the compression level (0-9).

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuemetadata_getCompressionStrategy

Description

Gets the compression strategy using which the messages sent on this queue will be compressed.

Function Signature

```
jint queuemetadata_getCompressionStrategy(FHandle qmd, jint strategy);
```

Parameters

`qmd` - The queue meta data object to operate on.

`strategy` - The compression strategy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuemetadata_getConfigParams

Description

Gets all the config params for this queue

Function Signature

```
jint queuemetadata_getConfigParams(FHandle qmd, FHandle ht);
```

Parameters

`qmd` - The queue meta data object to operate on.

`ht` - A hashtable object containing all the config parameters for this queue.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuemetadata_getKey

Description

Gets the key used to encrypt data.

Function Signature

```
jint queuemetadata_getKey(FHandle qmd, const char *key);
```

Parameters

`qmd` - the queue meta data object to operate on.

`key` - String encryption key.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuemetadata_getStorageType

Description

Gets the storage type for the queue.

Function Signature

```
jint queuemetadata_getStorageType(FHandle qmd, jbyte type);
```

Parameters

`qmd` - The queue meta data object to operate on.

`type` - Byte indicating the storage where all messages published on this queue are stored.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuemetadata_getStorageURL

Description

Gets the URL of the persistent storage (or it's context) where persistent information corresponding to this Queue is stored.

Function Signature

```
jint queuemetadata_getStorageURL(FHandle qmd, const char *url);
```

Parameters

`qmd` - The queue meta data object to operate on.

`Url` - URL of the persistent storage.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuemetadata_getType

Description

Gets the type of the metadata object

Function Signature

```
jint queuemetadata_getType(FHandle qmd, jbyte type);
```

Parameters

`qmd` - The queue metadata object to operate on.

`type` - Returns a byte indicating that the given metadata is a queue.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuemetadata_isCompressionEnabled

Description

Gets the compression enabled property.

Function Signature

```
jint queuemetadata_isCompressionEnabled(FHandle qmd, jboolean result);
```

Parameters

`qmd` - The queue metadata object to operate on.

`result` - TRUE if message compression is enabled on this queue and FALSE if message compression is disabled.

Returns

`MQ_SUCCESS` if successful, MQ_FAILURE if error.

queuemetadata_isEncrypted

Description

Indicates whether the queue is an encrypted queue or not.

Function Signature

```
jint queuemetadata_isEncrypted(FHandle qmd, jboolean result);
```

Parameters

`qmd` - The queue metadata object to operate on.

`result` - Indicates whether the queue is an encrypted queue or not.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuemetadata_isTemporary

Description

Checks whether the queue is a temporary queue.

Function Signature

```
jint queuemetadata_isTemporary(FHandle qmd, jboolean result);
```

Parameters

`qmd` - The queue meta data object to operate on.

`result` - TRUE if queue is a temporary queue, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuemetadata_setConfigParams

Description

Sets the configuration parameters.

Function Signature

```
jint queuemetadata_setConfigParams(FHandle qmd, FHandle ht);
```

Parameters

`qmd` - The queue metadata object to operate on.

`ht` - hashtable containing all the config parameters.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuemetadata_setEncryption

Description

Sets default encryption parameters in the queue.

Function Signature

```
jint queuemetadata_setEncryption(FHandle qmd);
```

Parameters

`qmd` - The queue meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuemetadata_setEncryption_1

Description

Sets encryption parameters in the queue.

Function Signature

```
jint queuemetadata_setEncryption_1(FHandle qmd, const char algo,
const char key);
```

Parameters

`qmd` - The queue metadata object to operate on.

`algo` - Algorithm used to encrypt data published on this queue.

`key` - Key used to encrypt data.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuemetadata_setStorageType

Description

Sets the storage type for the queue.

Function Signature

```
jint queuemetadata_setStorageType(FHandle qmd, jbyte type);
```

Parameters

`qmd` - The queue metadata object to operate on.

`type` - Byte representing the storage type where all messages published on this queue are stored.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuemetadata_setStorageURL

Description

Sets the URL of the persistent storage (or it's context) where persistent information corresponding to this Queue is stored.

Function Signature

```
jint queuemetadata_setStorageURL(FHandle qmd, const char strStor-
ageURL);
```

Parameters

`qmd` - The queue meta data object to operate on.

`strStorageURL` - The storage URL for the queue

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuemetadata_setTemporary

Description

Sets a boolean in the queue indicating that the queue is a temporary queue.

Function Signature

```
jint queuemetadata_setTemporary(FHandle qmd, jboolean bIsTempo-
rary);
```

Parameters

`qmd` - The queue meta data object to operate on.

`bIsTemporary` - TRUE if the queue is a temporary queue, FALSE otherwise.

Returns

`MQ_SUCCESS` if successful, MQ_FAILURE if error.

queuemetadata_validateMetaData

Description

Validates the parameters specified in the metadata.

Function Signature

```
jint queuemetadata_validateMetaData(FHandle qmd);
```

Parameters

`qmd` - The queue metadata object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuemetadata_getDescription

Description

Gets the description of the Administered Object The object implements the read-External method to restore its contents by calling the methods of DataInput for primitive types and readObject for objects, strings and arrays.

Function Signature

```
jint queuemetadata_getDescription(FHandle jmd, const char *description);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`description` - A description of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuemetadata_getName

Description

Gets the Name of the Administered Object.

Function Signature

```
jint queuemetadata_getName(FHandle jmd, const char *name);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`name` - The name of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuemetadata_hashCode

Description

Returns a hash code value for the object.

Function Signature

```
jint queuemetadata_hashCode(FHandle jmd, jint hc);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`hc` - The has code value of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuemetadata_setDescription

Description

Sets the description of the Metadata Object

Function Signature

```
jint queuemetadata_setDescription(FHandle jmd, const char strDe-
scription);
```

Parameters

`jmd` - The JMS metadata object to operate on.

`strDescription` - A description for this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuemetadata_setName

Description

Sets the Name of this MetaData Object

Function Signature

```
jint queuemetadata_setName(FHandle jmd, const char strName);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`strName` - The name of this meta data object.

Returns

`MQ_SUCCESS` if successful, MQ_FAILURE if error.

TopicMetaData

This class represents the MetaData information for a Topic as it is stored in an LDAP directory. Objects of this class are used to create Topics.

createTopicMetaData

Description

Creates a new topic meta data object.

Function Signature

```
jint createTopicMetaData(FHandle tmd);
```

Parameters

`tmd -` A newly created topic meta data object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicmetadata_canNPMessagesBeDropped

Description

This method returns the boolean specifying whether non-persistent messages published on this topic can be dropped or not.

Function Signature

```
jint topicmetadata_canNPMessagesBeDropped(FHandle tmd, jboolean re-
sult);
```

Parameters

`tmd` - The topic meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicmetadata_disableMessageDropping

Description

This method configures the topic to never drop the non-persistent messages.

Function Signature

```
jint topicmetadata_disableMessageDropping(FHandle tmd);
```

Parameters

`tmd` - The topic metadata object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicmetadata_enableCompression

Description

Enables message compression on this topic using the default level and strategy.

Function Signature

```
jint topicmetadata_enableCompression(FHandle tmd);
```

Parameters

`tmd` - The topic metadata object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicmetadata_enableCompression_1

Description

Enablesmessage compression on this topic using the level and strategy passed as parameters This would mean that all messages published on this topic will be compressed using the level and strategy specified in this API.

Function Signature

```
jint topicmetadata_enableCompression_1(FHandle tmd, jint level, jint
strategy);
```

Parameters

`tmd` - The topic metadata object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicmetadata_equals

Description

Compares the given object with this object.

Function Signature

```
jint topicmetadata_equals(FHandle tmd, FHandle obj, jboolean re-
sult);
```

Parameters

`tmd` - The topic meta data object to operate on.

`obj` - The object to compare against.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicmetadata_getAlgo

Description

Gets the algorithm used to encrypt data published on this topic

Function Signature

```
jint topicmetadata_getAlgo(FHandle tmd, const char *algo);
```

Parameters

`tmd` - The topic metadata object to operate on.

`algo` - Contains the name of the string encryption algorithm when the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicmetadata_getCompressionLevel

Description

Gets the compression level using which the messages sent on this Topic will be compressed.

Function Signature

```
jint topicmetadata_getCompressionLevel(FHandle tmd, jint level);
```

Parameters

`tmd` - The topic metadata object to operate on.

`level` - The compression level.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicmetadata_getCompressionStrategy

Description

Gets the compression strategy using which the messages sent on this topic will be compressed.

Function Signature

```
jint topicmetadata_getCompressionStrategy(FHandle tmd, jint strate-
gy);
```

Parameters

`tmd` - The topic metadata object to operate on.

`strategy` - The compression strategy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicmetadata_getConfigParams

Description

Gets all the config params for this topic

Function Signature

```
jint topicmetadata_getConfigParams(FHandle tmd, FHandle ht);
```

Parameters

`tmd` - The topic metadata object to operate on.

`ht` - A hashtable object containing all the configuration parameters for this topic.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicmetadata_getKey

Description

Gets the key used to encrypt data

Function Signature

```
jint topicmetadata_getKey(FHandle tmd, const char *key);
```

Parameters

`tmd` - The topic metadata object to operate on.

`key` - String encryption key.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicmetadata_getRetryIntervalForPublishing

Description

This method returns the Retry publish interval which specifies the values after which client retries to publish the NP messages.

Function Signature

```
jint topicmetadata_getRetryIntervalForPublishing(FHandle tmd, jint
retryInterval);
```

Parameters

`tmd` - The topic metadata object to operate on.

`interval` - Contains the retry interval after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicmetadata_getStorageType

Description

Gets the storage type for the topic.

Function Signature

```
jint topicmetadata_getStorageType(FHandle tmd, jbyte type);
```

Parameters

`tmd` - The topic metadata object to operate on.

`type` - Byte indicating the storage where all messages published on this topic are stored

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicmetadata_getStorageURL

Description

Gets the URL of the persistent storage (or it's context) where persistent information corresponding to this Topic is stored.

Function Signature

```
jint topicmetadata_getStorageURL(FHandle tmd, const char *url);
```

Parameters

`tmd` - The topic metadata object to operate on.

`Url` - Contains the URL of the persistent storage after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicmetadata_getType

Description

Gets the type of the metadata object

Function Signature

```
jint topicmetadata_getType(FHandle tmd, jbyte type);
```

Parameters

`tmd` - The topic metadata object to operate on.

`type` - Contains a byte indicating that the given metadata is a topic after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicmetadata_isCompressionEnabled

Description

Gets the compression enabled property.

Function Signature

```
jint topicmetadata_isCompressionEnabled(FHandle tmd, jboolean re-
sult);
```

Parameters

`tmd` - The topic metadata object to operate on.

`result` - TRUE if message compression is enabled on this topic and FALSE if message compression is disabled.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicmetadata_isEncrypted

Description

Indicates whether the topic is an encrypted topic or not.

Function Signature

```
jint topicmetadata_isEncrypted(FHandle tmd, jboolean result);
```

Parameters

`tmd` - The topic metadata object to operate on.

`result` - Whether the topic is an encrypted queue or not.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicmetadata_isTemporary

Description

Checks whether the topic is a temporary topic.

Function Signature

```
jint topicmetadata_isTemporary(FHandle tmd, jboolean result);
```

Parameters

`tmd` - The topic metadata object to operate on.

`result` - TRUE if topic is a Temporary Topic, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicmetadata_setConfigParams

Description

Sets the configuration parameters.

Function Signature

```
jint topicmetadata_setConfigParams(FHandle tmd, FHandle table);
```

Parameters

`tmd` - The topic metadata object to operate on.

`table` - A hashtable containing all the config parameters.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicmetadata_setEncryption

Description

Sets default encryption parameters in the topic.

Function Signature

```
jint topicmetadata_setEncryption(FHandle tmd);
```

Parameters

`tmd` - The topic metadata object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicmetadata_setEncryption_1

Description

Sets encryption parameters in the topic.

Function Signature

```
jint topicmetadata_setEncryption_1(FHandle tmd, const char algo,
const char key);
```

Parameters

`tmd` - The topic metadata object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicmetadata_setRetryIntervalForPublishing

Description

This method configures the value, after which the client should retry to publish the NP message.

Function Signature

```
jint topicmetadata_setRetryIntervalForPublishing(FHandle tmd, jint
interval);
```

Parameters

`tmd` - The topic metadata object to operate on.

`interval` - Specifies the value after which client should make the publish call.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicmetadata_setStorageType

Description

Sets the storage type for the topic.

Function Signature

```
jint topicmetadata_setStorageType(FHandle tmd, jbyte type);
```

Parameters

`tmd` - The topic metadata object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicmetadata_setStorageURL

Description

Sets the URL of the persistent storage (or it's context) where persistent information corresponding to this Topic is stored.

Function Signature

```
jint topicmetadata_setStorageURL(FHandle tmd, const char strStor-
ageURL);
```

Parameters

`tmd` - The topic metadata object to operate on.

`strStorageURL` - Storage URL for this topic.

Returns

`MQ_SUCCESS` if successful, MQ_FAILURE if error.

topicmetadata_setTemporary

Description

Sets a boolean in the topic indicating that the topic is a temporary topic.

Function Signature

```
jint topicmetadata_setTemporary(FHandle tmd, jboolean bIsTempo-
rary);
```

Parameters

`tmd` - The topic metadata object to operate on.

`bIsTemporary` - TRUE if the topic is a temporary topic, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicmetadata_validateMetaData

Description

Validates the parameters specified in the metadata.

Function Signature

```
jint topicmetadata_validateMetaData(FHandle tmd);
```

Parameters

`tmd` - The topic metadata object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicmetadata_getDescription

Description

Gets the description of the Administered Object The object implements the read-External method to restore its contents by calling the methods of DataInput for primitive types and readObject for objects, strings and arrays.

Function Signature

```
jint topicmetadata_getDescription(FHandle jmd, const char *descrip-
tion);
```

Parameters

`jmd` - The JMS metadata object to operate on.

`description` - A description of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicmetadata_getName

Description

Gets the Name of the Administered Object.

Function Signature

```
jint topicmetadata_getName(FHandle jmd, const char *name);
```

Parameters

`jmd` - The JMS metadata object to operate on.

`name` - The name of this object.

Returns

`MQ_SUCCESS` if successful, MQ_FAILURE if error.

topicmetadata_hashCode

Description

Returns a hash code value for the object.

Function Signature

```
jint topicmetadata_hashCode(FHandle jmd, jint hc);
```

Parameters

`jmd` - The JMS metadata object to operate on.

`hc` - The has code value of this object.

Returns

`MQ_SUCCESS` if successful, MQ_FAILURE if error.

topicmetadata_setDescription

Description

Sets the description of the Metadata Object

Function Signature

```
jint topicmetadata_setDescription(FHandle jmd, const char strDescription);
```

Parameters

`jmd` - The JMS metadata object to operate on.

`strDescription` - A description for this object.

Returns

`MQ_SUCCESS` if successful, MQ_FAILURE if error.

topicmetadata_setName

Description

Sets the Name of this MetaData Object

Function Signature

```
jint topicmetadata_setName(FHandle jmd, const char strName);
```

Parameters

`jmd` - The JMS metadata object to operate on.

`strName` - The name of this meta data object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ConnectionFactoryMetaData

This class represents the MetaData information for a ConnectionFactory at it is stored in a naming repository. This is the base class for all types of ConnectionFactory metadata objects and is used for creating all types (topic,queue,unified,admin) of ConnectionFactories in the FioranoMQ server.

createConnectionFactoryMetaData

Description

creates connection factory meta data object

Function Signature

```
jint createConnectionFactoryMetaData(FHandleFunction Signa-
turecfmd);
```

ParametersFunction Signature

`cfmd` - A newly created connection factory meta data object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_allowAutoRevalidation

Description

Sets the value for auto-revalidation.

Function Signature

```
jint cfmetadata_allowAutoRevalidation(FHandle cfmd, const charFunc-
tion SignatureallowAutoRevalidation);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`allowAutoRevalidationtrue -` If auto-revalidation is required, false otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_allowDurableConnections

Description

Enables durable connections for this ConnectionFactory.

Function Signature

```
jint cfmetadata_allowDurableConnections(FHandle cfmd, jboolean
allowDurableConnections);
```

Parameters

`cfmd` - the connection factory meta data object to operate on.

`allowDurableConnections` - TRUE if durable connections have to be enabled and FALSE otherwise

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_areDurableConnectionsAllowed

Description

Gives the status whether durable connection are allowed or not for this connection factory.

Function Signature

```
jint cfmetadata_areDurableConnectionsAllowed(FHandle cfmd, jbool-
eanFunction Signature result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if durable connections are allowed, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_disableCSPStoredMessageSend

Description

Disables the sending of any pending messages in the client side store of all durable connections created using this ConnectionFactory.

Function Signature

```
jint cfmetadata_disableCSPStoredMessageSend(FHandle cfmd, const
char dontSend);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`dontSend` - True if send stored messages has to be disabled else false.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_disablePing

Description

Disables the ping functionality for this ConnectionFactory.

Function Signature

```
jint cfmetadata_disablePing(FHandle cfmd, jboolean ping);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`ping` - TRUE will disabel pinging, FALSE will disable it.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_equals

Description

Compares the given object with this object.

Function Signature

```
jint cfmetadata_equals(FHandle cfmd, FHandle obj, jboolean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`obj` - The FHandle to compare against.

`result` - Will contain TRUE if the two objects are equal, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getAdminConnectionReconnectInterval

Description

Returns the value for the reconnect interval between two reconnect attempts made to all durable connections created using this Connection Factory.

Function Signature

```
jint cfmetadata_getAdminConnectionReconnectInterval(FHandle cfmd,
jlong interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - Will contain the reconnect interval after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getAutoDispatch

Description

Returns the value of the auto dispatch boolean for this Connection Factory.

Function Signature

```
jint cfmetadata_getAutoDispatch(FHandle cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the value of the auto dispatch boolean as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getBackupConnectURL

Description

Returns the back up connect URL at position num.

Function Signature

```
jint cfmetadata_getBackupConnectURL(FHandle cfmd, jint num, const
char *url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the backup URL.

`Url` - Will contain backup connection URL for this Connection Factory at position num.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getBackupInetAddress

Description

Gets the InetAddress at position num of the BackURL list.

Function Signature

```
jint cfmetadata_getBackupInetAddress(FHandle cfmd, jint num, FHandle inetAddress);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the InetAddress.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getBackupPort

Description

Gets the port for URL at position num of the BackURL list.

Function Signature

```
jint cfmetadata_getBackupPort(FHandle cfmd, jint num, jint backupPort);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the URL.

`backupPort` - Will contain the backup port number after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getBackupURLStrings

Description

Gets the semi colon seperated string of Backup URLs.

Function Signature

```
jint cfmetadata_getBackupURLStrings(FHandle cfmd, const char *url-
String);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`urlString` - Will contain the semicolon separated list of backup URLs.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getBatchTimeoutInterval

Description

Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint cfmetadata_getBatchTimeoutInterval(FHandle cfmd, const char
*interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - Will contain the CSP syncing delay.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getCBREnabled

Description

Returns TRUE if Content Based Routing is enabled for this Connection Factory and false otherwise.

Function Signature

```
jint cfmetadata_getCBREnabled(FHandle cfmd, const char *cbrEn-
abled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`cbrEnabled` - Will contain true if CBR is enabled else false.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getClientProxyURL

Description

Returns Client Proxy URL being used for TCP based connections created using this ConnectionFactory.

Function Signature

```
jint cfmetadata_getClientProxyURL(FHandle cfmd, const char *url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`url` - Will contain the client proxy URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getCompressionManager

Description

Gets the CompressionManager class name to be used for MessageCompression purposes.

Function Signature

```
jint cfmetadata_getCompressionManager(FHandle cfmd, const char
*compressionManager);
```

Parameters

`cfmd` - Tthe connection factory meta data object to operate on.

`compressionManager` - Will contain the name of the compression manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getConnectionClientID

Description

Returns the ClientID for this MetaData

Function Signature

```
jint cfmetadata_getConnectionClientID(FHandle cfmd, const char
*clientID);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`clientID` - Will contain the client ID.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getConnectURL

Description

Returns the URL of the FMQ server to which all connections created using this ConnectionFactory will be made.

Function Signature

```
jint cfmetadata_getConnectURL(FHandle cfmd, const char *url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`url` - Will contain the URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getCreateLocalSocket

Description

Gets the value of the local socket boolean.

Function Signature

```
jint cfmetadata_getCreateLocalSocket(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain true or false as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getCSPUpdateFrequency

Description

Gets the UpdateFrequency set for this connection factory.

Function Signature

```
jint cfmetadata_getCSPUpdateFrequency(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the update frequency as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getDurableConnectionBaseDir

Description

Gets the Durable Connection base directory for this ConnectionFactory as a string.

Function Signature

```
jint cfmetadata_getDurableConnectionBaseDir(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Path of the base directory that is being used for client side storage of messages.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getDurableConnectionReconnectInterval

Description

Returns the value for the reconnect interval between two reconnect attempts made to all durable connections created using this Connection Factory.

Function Signature

```
jint cfmetadata_getDurableConnectionReconnectInterval(FHandle
cfmd, jlong interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - Will contain the reconnect interval.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getFactoryMetadataParams

Description

Returns a Hashtable of all the properties for this ConnectionFactory.

Function Signature

```
jint cfmetadata_getFactoryMetadataParams(FHandle cfmd, FHandle ht);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`ht` - Will contain a hashtable of all the properties for the connection factory.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getHTTPProxyURL

Description

Returns HTTP Proxy URL being used for HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint cfmetadata_getHTTPProxyURL(FHandle cfmd, const char *url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`url` - Will contain the URL of the HTTP proxy as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getInetAddress

Description

Gets the Inet Address of the Server on which the ConnectionFactory resides in the Server.

Function Signature

```
jint cfmetadata_getInetAddress(FHandle cfmd, FHandle inetAddress);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`inetAddress` - Will contain the Inet address of the server.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getLMSEnabled

Description

Returns whether support for large messages is enabled or not

Function Signature

```
jint cfmetadata_getLMSEnabled(FHandle cfmd, jboolean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if large message support is enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getLookUpPreferredServer

Description

Gets the value for the lookup preferred server boolean.

Function Signature

```
jint cfmetadata_getLookUpPreferredServer(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain true or false as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getMaxAdminConnectionReconnectAttempts

Description

Returns the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server)that are made by a admin connection created using this Connection Factory, as string.

Function Signature

```
jint cfmetadata_getMaxAdminConnectionReconnectAttempts(FHandle
cfmd, jint result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the maximum number of reconnect attempts.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getMaxDurableConnectionReconnectAttempts

Description

Returns the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server)that are made by a durable connection created using this Connection Factory, as string.

Function Signature

```
jint cfmetadata_getMaxDurableConnectionReconnectAttempts(FHandle
cfmd, jint result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the maximum number of reconnect attempts.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getMaxSocketCreationTries

Description

Returns the value for the Maximum socket creation tries for all connections] made using this ConnectionFactory.

Function Signature

```
jint cfmetadata_getMaxSocketCreationTries(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the maximum number of socket creation tries as string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getPort

Description

Gets the port of the Server on which the ConnectionFactory resides in the Server.

Function Signature

```
jint cfmetadata_getPort(FHandle cfmd, jint result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the port number.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getProxyAuthenticationRealm

Description

Returns the Proxy Authentication realm being used on the proxy for HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint cfmetadata_getProxyAuthenticationRealm(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain name of the proxy authentication realm being used, as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getProxyCredentials

Description

Returns the password for the proxy through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint cfmetadata_getProxyCredentials(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the password of the proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getProxyPrincipal

Description

Returns the username for the proxy through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint cfmetadata_getProxyPrincipal(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the username of the proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getProxyType

Description

Returns the proxy name through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint cfmetadata_getProxyType(FHandle cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the name of the proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getPublishBehaviourInAutoRevalidation

Description

Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint cfmetadata_getPublishBehaviourInAutoRevalidation(FHandle
cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the CSP sync-up delay as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getPublishWaitDuringCSPSyncp

Description

Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint cfmetadata_getPublishWaitDuringCSPSyncp(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the CSP sync-up delay.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getSecurityManager

Description

Returns the SecurityManager for this ConnectionFactory. Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint cfmetadata_getSecurityManager(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the name of the security manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getSecurityProtocol

Description

Returns the security protocol for this ConnectionFactory.

Function Signature

```
jint cfmetadata_getSecurityProtocol(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the security protocol.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getServerProxyURL

Description

Returns Server Proxy URL being used for TCP based connections created using this ConnectionFactory.

Function Signature

```
jint cfmetadata_getServerProxyURL(FHandle cfmd, const char*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the URL of the server proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getShutdownHookEnabled

Description

Returns whether shutdown hook is enabled or not.

Function Signature

```
jint cfmetadata_getShutdownHookEnabled(FHandle cfmd, jboolean
result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if shutdown hook is enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getSleepSocketCreationTries

Description

Returns the value for the sleep time between two socket creation tries for all connections made using this ConnectionFactory.

Function Signature

```
jint cfmetadata_getSleepSocketCreationTries(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the sleep time bewteen socket creation tries as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getSocketTimeout

Description

Returns the timeout value for the sockets of all connections created using this ConnectionFactory.

Function Signature

```
jint cfmetadata_getSocketTimeout(FHandle cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the timeout value as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getSOCKSProxyURL

Description

Returns SOCKS Proxy URL being used for HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint cfmetadata_getSOCKSProxyURL(FHandle cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the URI of the SOCKS proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getTCPBatchSize

Description

Gets the message batch size for this ConnectionFactory.

Function Signature

```
jint cfmetadata_getTCPBatchSize(FHandle cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the batch size as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getTransportProtocol

Description

Returns the Transport Protocol for this ConnectionFactory.

Function Signature

```
jint cfmetadata_getTransportProtocol(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the transport protocol being used as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getXASocketTimeout

Description

Returns the timeout value for the socket used for the XAResource prepare and commit calls.

Function Signature

```
jint cfmetadata_getXASocketTimeout(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the timeout value as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_isAutoRevalidationEnabled

Description

Returns whether auto-revalidation is enabled or not.

Function Signature

```
jint cfmetadata_isAutoRevalidationEnabled(FHandle cfmd, const
char*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if auto-revalidation is enabled, false otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_isConnectURLUpdationAllowed

Description

Indicates whether Connect URL can be updated or not.

Function Signature

```
jint cfmetadata_isConnectURLUpdationAllowed(FHandle cfmd, jboolean
result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain a boolean indicating whether connect URL can be updated or not.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_isCSPStoredMessageSendDisabled

Description

Checks if send of pending messages is disabled for this connection factory.

Function Signature

```
jint cfmetadata_isCSPStoredMessageSendDisabled(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - True if send pending is disabled else false; null is returned if no value was set.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_isPingDisabled

Description

Checks if ping is enabled or disabled for this ConnectionFactory.

Function Signature

```
jint cfmetadata_isPingDisabled(FHandle cfmd, jboolean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if ping was disabled while creating the connection factory and FALSE if otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_isThreadContextClassLoaderUsed

Description

Gives value of USE_THREAD_CONTEXT_CLASS_LOADER.

Function Signature

```
jint cfmetadata_isThreadContextClassLoaderUsed(FHandle cfmd, jbool-
ean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain boolean specifying whether USE_THREAD_CONTEXT_CLASS_LOADER is true or false.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_loadEnvParams

Description

Loads the properties from the Env that are not already specified in the metaData.

Function Signature

```
jint cfmetadata_loadEnvParams(FHandle cfmd, FHandle env);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`env` - The Env properties that need to be loaded into the metaData.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setAdminConnectionReconnectInterval

Description

Sets the value for the reconnect interval which is the interval between two successive reconnect attempts that would be made by a Durable connection created using this Connection Factory.

Function Signature

```
jint cfmetadata_setAdminConnectionReconnectInterval(FHandle cfmd,
const char interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - The reconnect interval as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setAutoDispatch

Description

Sets the value for the auto-dispatch boolean.

Function Signature

```
jint cfmetadata_setAutoDispatch(FHandle cfmd, const char autoDis-
patch);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`autoDispatch` - True or false passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setBackupConnectURL

Description

Sets the back up connect URL at position num for this Connection Factory.

Function Signature

```
jint cfmetadata_setBackupConnectURL(FHandle cfmd, jint num, const
char strBackupConnectURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the backup URL (0 for first backup URL).

`strBackupURL` - The backup URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setBackupConnectURLs

Description

Sets the Backup URLs.

Function Signature

```
jint cfmetadata_setBackupConnectURLs(FHandle cfmd, const char back-
upUrls);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`backupUrls` - A semicolon seperated list of URLs.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setBatchTimeoutInterval

Description

Sets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint cfmetadata_setBatchTimeoutInterval(FHandle cfmd, const char
batchTimeoutInterval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`batchTimeoutInterval` - The delay in publishing new messages when CSP is syncing up.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setCBREnabled

Description

Enables Content Based Routing for all connections that will be created using this ConnectionFactory.

Function Signature

```
jint cfmetadata_setCBREnabled(FHandle cfmd, const char isCBREn-
abled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isCBREnabled` - True to enable Content Based Routing else false, passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setClientProxyURL

Description

Sets the URL for the Client side Proxy through which TCP based Connections created using this ConnectionFactory will be routed using HTTP Tunneling.

Function Signature

```
jint cfmetadata_setClientProxyURL(FHandle cfmd, const char prox-
yURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The client proxy URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setCompressionManager

Description

Sets the CompressionManager class name to be used for MessageCompresson purposes.

Function Signature

```
jint cfmetadata_setCompressionManager(FHandle cfmd, const char
nmanager);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`manager` - Name of the compression manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setConnectionClientID

Description

Sets the ClientID for this MetaData.

Function Signature

```
jint cfmetadata_setConnectionClientID(FHandle cfmd, const char cli-
entID);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`clientID` - The client ID.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setConnectURL

Description

Sets the primary Connect URL for this ConnectionFactory.

Function Signature

```
jint cfmetadata_setConnectURL(FHandle cfmd, const char strConnec-
tURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setCreateLocalSocket

Description

Sets the value for the local socket boolean.

Function Signature

```
jint cfmetadata_setCreateLocalSocket(FHandle cfmd, const char
localSocket);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`localSocket` - True of false as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setCSPUpdateFrequency

Description

Sets the UpdateFrequency to the value passed.

Function Signature

```
jint cfmetadata_setCSPUpdateFrequency(FHandle cfmd, const char fre-
quency);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`frequency` - The CSP update frequency.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setDurableConnectionBaseDir

Description

Sets the Durable Connection Base directory for this ConnectionFactory as a string.

Function Signature

```
jint cfmetadata_setDurableConnectionBaseDir(FHandle cfmd, const
char durBaseDir);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`durBaseDir` - Fully qualified path of the base directory for client side storage of messages

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setDurableConnectionReconnectInterval

Description

Sets the value for the reconnect interval which is the interval between two successive reconnect attempts that would be made by a Durable connection created using this Connection Factory.

Function Signature

```
jint cfmetadata_setDurableConnectionReconnectInterval(FHandle
cfmd, const char interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - The reconnect interval passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setFactoryMetadataParams

Description

Sets the Hashtable containing all properties for this Connection Factory.

Function Signature

```
jint cfmetadata_setFactoryMetadataParams(FHandle cfmd, FHandle
params);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`params` - Hashtable containing the name value pairs of all Connection Factory properties.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setHTTPProxyURL

Description

Sets the URL for the HTTP Proxy through which HTTP based Connections created using this ConnectionFactory will be routed.

Function Signature

```
jint cfmetadata_setHTTPProxyURL(FHandle cfmd, const char proxyURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The URL of the HTTP proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setLMSEnabled

Description

Enables/disables large message support.

Function Signature

```
jint cfmetadata_setLMSEnabled(FHandle cfmd, jboolean isLMSEnabled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isLMSEnabled` - TRUE if Large support has to be enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setLookUpPreferredServer

Description

Sets the value for the lookup preferred server boolean.

Function Signature

```
jint cfmetadata_setLookUpPreferredServer(FHandle cfmd, const char
lookupPreferred);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`lookupPreferred` - True or false passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setMaxAdminConnectionReconnectAttempts

Description

Sets the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server) that will be made by an admin connection created using this Connection Factory.

Function Signature

```
jint cfmetadata_setMaxAdminConnectionReconnectAttempts(FHandle
cfmd,const char numAttempts);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`numAttempts` - Maximum number of reconnect attempts passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setMaxDurableConnectionReconnectAttempts

Description

Sets the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server)that will be made by a durable connection created using this Connection Factory.

Function Signature

```
jint cfmetadata_setMaxDurableConnectionReconnectAttempts(FHandle cfmd, const char numAttempts);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`numAttempts` - Maximum number of reconnect attempts passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setMaxSocketCreationTries

Description

Sets the value for the maximum socket creation tries that the FioranoMQ runtime will make for creating the socket for all connections made using this Connection-Factory.

Function Signature

```
jint cfmetadata_setMaxSocketCreationTries(FHandle cfmd, const char socketTries);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`socketTries` - The number of socket creation tries passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setProxyAuthenticationRealm

Description

Sets the Authentication Realm being used on the Proxy through which HTTP based Connections created using this ConnectionFactory will be routed.

Function Signature

```
jint cfmetadata_setProxyAuthenticationRealm(FHandle cfmd, const
char realm);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`realm` - Name of the proxy authentication realm to be used.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setProxyCredentials

Description

Sets the password for validation at the proxy (if required) which is to be used for the HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint cfmetadata_setProxyCredentials(FHandle cfmd, const char pass-
word);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`password` - The password.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setProxyPrincipal

Description

Sets the username for the proxy (if required) through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint cfmetadata_setProxyPrincipal(FHandle cfmd, const char user-
name);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`username` - The username.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setProxyType

Description

Sets the type (name) of the proxy being used for the HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint cfmetadata_setProxyType(FHandle cfmd, const char proxy);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxy` - The type of the proxy as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setPublishBehaviourInAutoRevalidation

Description

Sets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint cfmetadata_setPublishBehaviourInAutoRevalidation(FHandle
cfmd, const char publishBehaviour);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`publishBehaviour` - Publisher behaviour on auto-revalidation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setPublishWaitDuringCSPSyncp

Description

Sets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint cfmetadata_setPublishWaitDuringCSPSyncp(FHandle cfmd, const
char delay);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`delay` - The delay passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setSecurityManager

Description

Sets the security manager that will be used to make SSL connections using this ConnectionFactory.

Function Signature

```
jint cfmetadata_setSecurityManager(FHandle cfmd, const charFunction
Signaturemanager);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`manager` - The name of the security manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setSecurityProtocol

Description

Sets the security protocol over which all connections created using this ConnectionFactory communicate with the FMQ server

Function Signature

```
jint cfmetadata_setSecurityProtocol(FHandle cfmd, const char protocol);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`protocol` - Security protocol, possible values are PHAOS_SSL and JSSE_SSL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setServerProxyURL

Description

Sets the URL for the Server side Proxy through which TCP based Connections created using this ConnectionFactory will be routed using HTTP Tunneling.

Function Signature

```
jint cfmetadata_setServerProxyURL(FHandle cfmd, const char proxyURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The URL of the server proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setShutdownHookEnabled

Description

Enables/disables shutdown hook.

Function Signature

```
jint cfmetadata_setShutdownHookEnabled(FHandle cfmd, jboolean
isShutdownHookEnabled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isShutdownHookEnabled` - TRUE if shutdown hook had to be enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setSleepSocketCreationTries

Description

Sets the value for the sleep time bewteen two creation tries that the FioranoMQ runtime. These two creation tries are made for creating the socket for all connections made using this ConnectionFactory.

Function Signature

```
jint cfmetadata_setSleepSocketCreationTries(FHandle cfmd, const
char sleepTime);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setSocketTimeout

Description

Sets the timeout value for the sockets of all connections created using this ConnectionFactory.

Function Signature

```
jint cfmetadata_setSocketTimeout(FHandle cfmd, const char timeout);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`timeout` - The timeout value passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setSOCKSProxyURL

Description

Sets the URL for the SOCKS Proxy through which HTTP based Connections created using this ConnectionFactory will be routed.

Function Signature

```
jint cfmetadata_setSOCKSProxyURL(FHandle cfmd, const char proxyURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The URL of the SOCKS proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setTCPBatchSize

Description

Sets the value of the message batch size that will be used for NP messages being sent using all connections created using this Connection Factory.

Function Signature

```
jint cfmetadata_setTCPBatchSize(FHandle cfmd, const char batchSize);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`batchSize` - The message batch size that will be used for NP messages.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setTransportProtocol

Description

Sets the transport protocol over which all connections created using this ConnectionFactory communicate with the FMQ server.

Function Signature

```
jint cfmetadata_setTransportProtocol(FHandle cfmd, const char protocol);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`protocol` - The protocol. Possible values are TCP or HTTP.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_setXASocketTimeout

Description

Sets the timeout value for the socket when the XAResource prepare or commit call is made from the runtime.

Function Signature

```
jint cfmetadata_setXASocketTimeout(FHandle cfmd, const char timeout);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`timeout` - The timeout value as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_updateConnectURL

Description

Sets a boolean in ConnectionFactory indicating whether Connect URL can be updated or not

Function Signature

```
jint cfmetadata_updateConnectURL(FHandle cfmd, jboolean flag);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`flag` - Indicates whether connect URL can be updated or not.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_useThreadContextClassLoader

Description

Sets the value of USE_THREAD_CONTEXT_CLASS_LOADER

Function Signature

```
jint cfmetadata_useThreadContextClassLoader(FHandle cfmd, jboolean
isThreadContextClassLoaderUsed);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isThreadContextClassLoaderUsed` - TRUE or FALSE.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_validateMetaData

Description

Validates if the URL and the name have been specified properly or not.

Function Signature

```
jint cfmetadata_validateMetaData(FHandle cfmd);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_validateURL

Description

Validates the URL to see if the URL is an IP address/hostname. It replaces "local-host" with complete IP address.

Function Signature

```
jint cfmetadata_validateURL(FHandle cfmd);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

AdminConnectionFactoryMetaData

This class represents the MetaData information for an AdminConnectionFactory as it is stored in an LDAP directory. This class is required for making admin connection to the server.

createAdminConnectionFactoryMetaData

Description

Creates a new AdminConnectionFactoryMetaData object.

Function Signature

```
jint createAdminConnectionFactoryMetaData(FHandle acfmd);
```

Parameters

`cfmd` - A newly created AdminConnectionFactoryMetaData object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_equals

Description

Compares two AdminConnectionFactoryMetaData objects for equality.

Function Signature

```
jint acfmetadata_equals(FHandle acfmd, FHandle obj, jboolean
result);
```

Parameters

`acfmd` - The AdminConnectionFactoryMetaData object to operate on.

`result` - Will contain TRUE if the two objects are equal, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_allowAutoRevalidation

Description

Sets the value for auto-revalidation.

Function Signature

```
jint acfmetadata_allowAutoRevalidation(FHandle cfmd, const char
allowAutoRevalidation);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`allowAutoRevalidationtrue` - If auto-revalidation is required, false otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_allowDurableConnections

Description

Enablesthe durable connections for this ConnectionFactory.

Function Signature

```
jint acfmetadata_allowDurableConnections(FHandle cfmd, jboolean
allowDurableConnections);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`allowDurableConnections` - TRUE if durable connections have to be enabled and FALSE otherwise

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_areDurableConnectionsAllowed

Description

Gives the status whether durable connection are allowed or not for this connection factory.

Function Signature

```
jint acfmetadata_areDurableConnectionsAllowed(FHandle cfmd, jbool-
ean allowDurableConnections);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if durable connections are allowed, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_disableCSPStoredMessageSend

Description

Disables the sending of any pending messages in the client side store of all durable connections created using this ConnectionFactory.

Function Signature

```
jint acfmetadata_disableCSPStoredMessageSend(FHandle cfmd, const
charFunction Signature dontSend);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`dontSend` - True if send stored messages has to be disabled else false

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_disablePing

Description

Disables the ping functionality for this ConnectionFactory.

Function Signature

```
jint acfmetadata_disablePing (FHandle cfmd, jboolean ping);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`ping` - TRUE will disabel pinging, FALSE will disable it.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getAdminConnectionReconnectInterval

Description

Returns the value for the reconnect interval between two reconnect attempts made all durable connections created using this Connection Factory.

Function Signature

```
jint acfmetadata_getAdminConnectionReconnectInterval(FHandle cfmd,
jlong interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - Will contain the reconnect interval after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getAutoDispatch

Description

Returns the value of the auto dispatch boolean for this Connection Factory.

Function Signature

```
jint acfmetadata_getAutoDispatch(FHandle cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the value of the auto dispatch boolean as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getBackupConnectURL

Description

Returns the back up connect URL at position num.

Function Signature

```
jint acfmetadata_getBackupConnectURL(FHandle cfmd, jint num, const
char *url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the backup URL.

`url` - Will contain backup connection URL for this Connection Factory at position num.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getBackupInetAddress

Description

Gets the InetAddress at position num of the BackURL list.

Function Signature

```
jint acfmetadata_getBackupInetAddress(FHandle cfmd, jint num, FHan-
dle inetAddress)
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the InetAddress.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getBackupPort

Description

Gets the port for URL at position num of the BackURL list.

Function Signature

```
jint acfmetadata_getBackupPort(FHandle cfmd, jint num, jint backup-
Port);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the URL.

`backupPort` - Will contain the backup port number after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getBackupURLStrings

Description

Gets the semicolon seperated string of Backup URLs.

Function Signature

```
jint acfmetadata_getBackupURLStrings(FHandle cfmd, const char *url-
String);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`urlString` - Will contain the semicolon seperated list of backup URLs.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getBatchTimeoutInterval

Description

Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint acfmetadata_getBatchTimeoutInterval(FHandle cfmd, const char
*interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - Will contain the CSP syncing delay.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getCBREnabled

Description

Returns true if Content Based Routing is enabled for this Connection Factory and false otherwise.

Function Signature

```
jint acfmetadata_getCBREnabled(FHandle cfmd, const char *cbrEn-
abled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`cbrEnabled` - Will contain true if CBR is enabled else false.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getClientProxyURL

Description

Returns Client Proxy URL being used for TCP based connections created using this ConnectionFactory.

Function Signature

```
jint acfmetadata_getClientProxyURL(FHandle cfmd, const char *url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`url` - will contain the client proxy URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getCompressionManager

Description

Gets the CompressionManager class name to be used for MessageCompression purposes.

Function Signature

```
jint acfmetadata_getCompressionManager(FHandle cfmd, const char
*compressionManager);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`compressionManager` - Will contain the name of the compression manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getConnectionClientID

Description

Returns the ClientID for this MetaData

Function Signature

```
jint acfmetadata_getConnectionClientID(FHandle cfmd, const char
*clientID);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getConnectURL

Description

Returns the URL of the FMQ server to which all connections created using this ConnectionFactory will be made.

Function Signature

```
jint acfmetadata_getConnectURL(FHandle cfmd, const char *url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`url` - Will contain the URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getCreateLocalSocket

Description

Gets the value of the local socket boolean.

Function Signature

```
jint acfmetadata_getCreateLocalSocket(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain true or false as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getCSPUpdateFrequency

Description

Gets the UpdateFrequency set for this connection factory.

Function Signature

```
jint acfmetadata_getCSPUpdateFrequency(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the update frequency as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getDurableConnectionBaseDir

Description

Gets the Durable Connection base directory for this ConnectionFactory as a string.

Function Signature

```
jint acfmetadata_getDurableConnectionBaseDir(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Path of the base directory that is being used for client side storage of messages.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getDurableConnectionReconnectInterval

Description

Returns the value for the reconnect interval between two reconnect attempts made all durable connections created using this Connection Factory.

Function Signature

```
jint acfmetadata_getDurableConnectionReconnectInterval(FHandle
cfmd, jlong interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - Will contain the reconnect interval.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getFactoryMetadataParams

Description

Returns a Hashtable of all the properties for this ConnectionFactory.

Function Signature

```
jint acfmetadata_getFactoryMetadataParams(FHandle cfmd, FHandle
ht);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`ht` - Will contain a hashtable of all the properties for the connection factory.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getHTTPProxyURL

Description

Returns HTTP Proxy URL being used for HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint acfmetadata_getHTTPProxyURL(FHandle cfmd, const char *url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`url` - Will contain the URL of the HTTP proxy as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getInetAddress

Description

Gets the Inet Address of the Server on which the ConnectionFactory resides in the Server.

Function Signature

```
jint acfmetadata_getInetAddress(FHandle cfmd, FHandle inetAddress);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`inetAddress` - Will contain the Inet address of the server.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getLMSEnabled

Description

Returns whether support for large messages is enabled or not

Function Signature

```
jint acfmetadata_getLMSEnabled(FHandle cfmd, jboolean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if large message support is enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getLookUpPreferredServer

Description

Gets the value for the lookup preferred server boolean.

Function Signature

```
jint acfmetadata_getLookUpPreferredServer(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain true or false as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getMaxAdminConnectionReconnectAttempts

Description

Returns the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server)that are made by a admin connection created using this Connection Factory, as string.

Function Signature

```
jint acfmetadata_getMaxAdminConnectionReconnectAttempts(FHandle
cfmd, jint result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the maximum number of reconnect attempts.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getMaxDurableConnectionReconnectAttempts

Description

Returns the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server)that are made by a durable connection created using this Connection Factory, as string.

Function Signature

```
jint acfmetadata_getMaxDurableConnectionReconnectAttempts(FHandle
cfmd, jint result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the maximum number of reconnect attempts.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getMaxSocketCreationTries

Description

Returns the value for the Maximum socket creation tries for all connections] made using this ConnectionFactory.

Function Signature

```
jint acfmetadata_getMaxSocketCreationTries(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the maximum number of socket creation tries as string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getPort

Description

Gets the port of the Server on which the ConnectionFactory resides in the Server.

Function Signature

```
jint acfmetadata_getPort(FHandle cfmd, jint result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the port number.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getProxyAuthenticationRealm

Description

Returns the Proxy Authentication realm being used on the proxy for HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint acfmetadata_getProxyAuthenticationRealm(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`Rrsult` - Will contain name of the proxy authentication realm being used, as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getProxyCredentials

Description

Returns the password for the proxy through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint acfmetadata_getProxyCredentials(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the password of the proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getProxyPrincipal

Description

Returns the username for the proxy through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint acfmetadata_getProxyPrincipal(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the username of the proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getProxyType

Description

Returns the proxy name through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint acfmetadata_getProxyType(FHandle cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the name of the proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getPublishBehaviourInAutoRevalidation

Description

Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint acfmetadata_getPublishBehaviourInAutoRevalidation(FHandle
cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the CSP sync-up delay as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getPublishWaitDuringCSPSyncp

Description

Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint acfmetadata_getPublishWaitDuringCSPSyncp(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the CSP sync-up delay.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getSecurityManager

Description

Returns the SecurityManager for this ConnectionFactory. Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint acfmetadata_getSecurityManager(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the name of the security manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getSecurityProtocol

Description

Returns the security protocol for this ConnectionFactory.

Function Signature

```
jint acfmetadata_getSecurityProtocol(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the security protocol.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getServerProxyURL

Description

Returns Server Proxy URL being used for TCP based connections created using this ConnectionFactory.

Function Signature

```
jint acfmetadata_getServerProxyURL(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the URL of the server proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getShutdownHookEnabled

Description

Returns whether shutdown hook is enabled or not.

Function Signature

```
jint acfmetadata_getShutdownHookEnabled(FHandle cfmd, jboolean
result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if shutdown hook is enabled, FALSE otherwise.

Returns

`MQ_SUCCESS` if successful, MQ_FAILURE if error.

acfmetadata_getSleepSocketCreationTries

Description

Returns the value for the sleep time between two socket creation tries for all connections] made using this ConnectionFactory.

Function Signature

```
jint acfmetadata_getSleepSocketCreationTries(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the sleep time bewteen socket creation tries as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getSocketTimeout

Description

Returns the timeout value for the sockets of all connections created using this ConnectionFactory.

Function Signature

```
jint acfmetadata_getSocketTimeout(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the timeout value as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getSOCKSProxyURL

Description

Returns SOCKS Proxy URL being used for HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint acfmetadata_getSOCKSProxyURL(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the URI of the SOCKS proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getTCPBatchSize

Description

Gets the message batch size for this ConnectionFactory.

Function Signature

```
jint acfmetadata_getTCPBatchSize(FHandle cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the batch size as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getTransportProtocol

Description

Returns the Transport Protocol for this ConnectionFactory.

Function Signature

```
jint acfmetadata_getTransportProtocol(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the transport protocol being used as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getXASocketTimeout

Description

Returns the timeout value for the socket used for the XAResource prepare and-commit calls.

Function Signature

```
jint acfmetadata_getXASocketTimeout(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the timeout value as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_isAutoRevalidationEnabled

Description

Returns whether auto-revalidation is enabled or not.

Function Signature

```
jint acfmetadata_isAutoRevalidationEnabled(FHandle cfmd, const
char*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain true if auto-revalidation is enabled, false otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_isConnectURLUpdationAllowed

Description

Indicates whether Connect URL can be updated or not.

Function Signature

```
jint acfmetadata_isConnectURLUpdationAllowed(FHandle cfmd, jboolean
result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain a boolean indicating whether connect URL can be updated or not.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_isCSPStoredMessageSendDisabled

Description

Checks if send of pending messages is disabled for this connection factory.

Function Signature

```
jint acfmetadata_isCSPStoredMessageSendDisabled(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - True if send pending is disabled else false; null is returned if no value was set.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_isPingDisabled

Description

Checks if ping is enabled or disabled for this ConnectionFactory.

Function Signature

```
jint acfmetadata_isPingDisabled(FHandle cfmd, jboolean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if ping was disabled while creating the connection factory and FALSE if otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_isThreadContextClassLoaderUsed

Description

Gives value of USE_THREAD_CONTEXT_CLASS_LOADER.

Function Signature

```
jint acfmetadata_isThreadContextClassLoaderUsed(FHandle cfmd,
jboolean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain boolean specifying whether USE_THREAD_CONTEXT_CLASS_LOADER is true or false.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_loadEnvParams

Description

Loads the properties from the Env that are not already specified in the metaData.

Function Signature

```
jint acfmetadata_loadEnvParams(FHandle cfmd, FHandle env);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`env` - The Env properties that need to be loaded into the metaData.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setAdminConnectionReconnectInterval

Description

Sets the value for the reconnect interval which is the interval between two successive reconnect attempts that would be made by a Durable connection created using this Connection Factory.

Function Signature

```
jint acfmetadata_setAdminConnectionReconnectInterval(FHandle cfmd,
const char interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - The reconnect interval as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setAutoDispatch

Description

Sets the value for the auto dispatch boolean.

Function Signature

```
jint acfmetadata_setAutoDispatch(FHandle cfmd, const char autoDis-
patch);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`autoDispatch` - True or false passed as a string.

Returns

`MQ_SUCCESS` if successful, MQ_FAILURE if error.

acfmetadata_setBackupConnectURL

Description

Sets the back up connect url at position num for this Connection Factory.

Function Signature

```
jint acfmetadata_setBackupConnectURL(FHandle cfmd, jint num, const
char strBackupConnectURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the backup URL (0 for first backup URL).

`strBackupURL` - The backup URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setBackupConnectURLs

Description

Sets the Backup URL's.

Function Signature

```
jint acfmetadata_setBackupConnectURLs(FHandle cfmd, const char
backupUrls);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`backupUrls` - A semicolon seperated list of URLs.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setBatchTimeoutInterval

Description

Sets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint acfmetadata_setBatchTimeoutInterval(FHandle cfmd, const char
batchTimeoutInterval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`batchTimeoutInterval` - The delay in publishing new messages when CSP is syncing up.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setCBREnabled

Description

Enable Content Based Routing for all connections that will be created using this ConnectionFactory.

Function Signature

```
jint acfmetadata_setCBREnabled(FHandle cfmd, const char isCBREn-
abled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isCBREnabled` - True to enable Content Based Routing else false, passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setClientProxyURL

Description

Sets the URL for the Client side Proxy through which TCP based Connections created using this ConnectionFactory will be routed using HTTP Tunneling.

Function Signature

```
jint acfmetadata_setClientProxyURL(FHandle cfmd, const char proxyURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The client proxy URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setCompressionManager

Description

Sets the CompressionManager class name to be used for MessageCompresson purposes.

Function Signature

```
jint acfmetadata_setCompressionManager(FHandle cfmd, const char nmanager);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`manager` - Name of the compression manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setConnectionClientID

Description

Sets the ClientID for this MetaData.

Function Signature

```
jint acfmetadata_setConnectionClientID(FHandle cfmd, const char
clientID);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`clientID` - The client ID.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setConnectURL

Description

Sets the primary Connect URL for this ConnectionFactory.

Function Signature

```
jint acfmetadata_setConnectURL(FHandle cfmd, const char strConnec-
tURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setCreateLocalSocket

Description

Sets the value for the local socket boolean.

Function Signature

```
jint acfmetadata_setCreateLocalSocket(FHandle cfmd, const char
localSocket);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`localSocket` - True of false as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setCSPUpdateFrequency

Description

Sets the UpdateFrequency to the value passed.

Function Signature

```
jint acfmetadata_setCSPUpdateFrequency(FHandle cfmd, const char
frequency);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`frequency` - The CSP update frequency.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setDurableConnectionBaseDir

Description

Sets the Durable Connection Base directory for this ConnectionFactory as a string.

Function Signature

```
jint acfmetadata_setDurableConnectionBaseDir(FHandle cfmd, const
char durBaseDir);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`durBaseDir` - Fully qualified path of the base directory for client side storage of messages

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setDurableConnectionReconnectInterval

Description

Sets the value for the reconnect interval which is the interval between two successive reconnect attempts that would be made by a Durable connection created using this Connection Factory.

Function Signature

```
jint acfmetadata_setDurableConnectionReconnectInterval(FHandle
cfmd, const char interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - The reconnect interval passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setFactoryMetadataParams

Description

Sets the Hashtable containing all properties for this Connection Factory.

Function Signature

```
jint acfmetadata_setFactoryMetadataParams(FHandle cfmd, FHandle
params);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`params` - Hashtable containing the name value pairs of all Connection Factory properties.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setHTTPProxyURL

Description

Sets the URL for the HTTP Proxy through which HTTP based Connections created using this ConnectionFactory will be routed.

Function Signature

```
jint acfmetadata_setHTTPProxyURL(FHandle cfmd, const char proxyURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The URL of the HTTP proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setLMSEnabled

Description

Enables/disables large message support.

Function Signature

```
jint acfmetadata_setLMSEnabled(FHandle cfmd, jboolean isLMSEnabled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isLMSEnabled` - TRUE if Large support has to be enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setLookUpPreferredServer

Description

Sets the value for the lookup preferred server boolean.

Function Signature

```
jint acfmetadata_setLookUpPreferredServer(FHandle cfmd, const char lookupPreferred);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`lookupPreferred` - True or false passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setMaxAdminConnectionReconnectAttempts

Description

Sets the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server) that will be made by an admin connection created using this Connection Factory.

Function Signature

```
jint acfmetadata_setMaxAdminConnectionReconnectAttempts(FHandle
cfmd,const char numAttempts);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`numAttempts` - Maximum number of reconnect attempts passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setMaxDurableConnectionReconnectAttempts

Description

Sets the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server) that will be made by a durable connection created using this Connection Factory.

Function Signature

```
jint acfmetadata_setMaxDurableConnectionReconnectAttempts(FHandle
cfmd, const char numAttempts);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`numAttempts` - Maximum number of reconnect attempts passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setMaxSocketCreationTries

Description

Sets the value for the maximum socket creation tries that the FioranoMQ runtime will make for creating the socket for all connections made using this Connection-Factory.

Function Signature

```
jint acfmetadata_setMaxSocketCreationTries(FHandle cfmd, const char socketTries);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`socketTries` - The number of socket creation tries passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setProxyAuthenticationRealm

Description

Sets the Authentication Realm being used on the Proxy through which HTTP based Connections created using this ConnectionFactory will be routed.

Function Signature

```
jint acfmetadata_setProxyAuthenticationRealm(FHandle cfmd, const char realm);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`realm` - Name of the proxy authentication realm to be used.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setProxyCredentials

Description

Sets the password for getting validated at the proxy (if required) which is to be used for the HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint acfmetadata_setProxyCredentials(FHandle cfmd, const char pass-
word);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`password` - The password.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setProxyPrincipal

Description

Sets the username for the proxy (if required) through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint acfmetadata_setProxyPrincipal(FHandle cfmd, const char user-
name);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`username` - The username.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setProxyType

Description

Sets the type (name) of the proxy being used for the HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint acfmetadata_setProxyType(FHandle cfmd, const char proxy);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxy` - The type of the proxy as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setPublishBehaviourInAutoRevalidation

Description

Sets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint acfmetadata_setPublishBehaviourInAutoRevalidation(FHandle
cfmd, const char publishBehaviour);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`publishBehaviour` - Publisher behaviour on auto-revalidation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setPublishWaitDuringCSPSyncp

Description

Sets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint acfmetadata_setPublishWaitDuringCSPSyncp(FHandle cfmd, const
char delay);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`delay` - The delay passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setSecurityManager

Description

Sets the security manager that will be used to make SSL connections using this ConnectionFactory.

Function Signature

```
jint acfmetadata_setSecurityManager(FHandle cfmd, const charFunc-
tion Signaturemanager);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`manager` - The name of the security manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setSecurityProtocol

Description

Sets the security protocol over which all connections created using this ConnectionFactory communicate with the FMQ server

Function Signature

```
jint acfmetadata_setSecurityProtocol(FHandle cfmd, const char pro-
tocol);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`protocol` - Security protocol, possible values are PHAOS_SSL and JSSE_SSL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setServerProxyURL

Description

Sets the URL for the Server side Proxy through which TCP based Connections created using this ConnectionFactory will be routed using HTTP Tunneling.

Function Signature

```
jint acfmetadata_setServerProxyURL(FHandle cfmd, const char prox-
yURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The URL of the server proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setShutdownHookEnabled

Description

Enables/disables shutdown hook.

Function Signature

```
jint acfmetadata_setShutdownHookEnabled(FHandle cfmd, jboolean
isShutdownHookEnabled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isShutdownHookEnabled` - TRUE if shutdown hook had to be enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setSleepSocketCreationTries

Description

Sets the value for the sleep time bewteen two creation tries that the FioranoMQ runtime will make for creating the socket for all connections made using this ConnectionFactory.

Function Signature

```
jint acfmetadata_setSleepSocketCreationTries(FHandle cfmd, const
char sleepTime);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setSocketTimeout

Description

Sets the timeout value for the sockets of all connections created using this ConnectionFactory.

Function Signature

```
jint acfmetadata_setSocketTimeout(FHandle cfmd, const char timeout);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`timeout` - The timeout value passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setSOCKSProxyURL

Description

Sets the URL for the SOCKS Proxy through which HTTP based Connections created using this ConnectionFactory will be routed.

Function Signature

```
jint acfmetadata_setSOCKSProxyURL(FHandle cfmd, const char proxyURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The URL of the SOCKS proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setTCPBatchSize

Description

Sets the value of the message batch size that will be used for NP messages being sent using all connections created using this Connection Factory.

Function Signature

```
jint acfmetadata_setTCPBatchSize(FHandle cfmd, const char batch-
Size);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`batchSize` - The message batch size that will be used for NP messages.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setTransportProtocol

Description

Sets the transport protocol over which all connections created using this ConnectionFactory communicate with the FMQ server.

Function Signature

```
jint acfmetadata_setTransportProtocol(FHandle cfmd, const char pro-
tocol);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`protocol` - The protocol. Possible values are TCP or HTTP.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setXASocketTimeout

Description

Sets the timeout value for the socket when the XAResource prepare or commit call is made from the runtime.

Function Signature

```
jint acfmetadata_setXASocketTimeout(FHandle cfmd, const char time-
out);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`timeout` - The timeout value as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_updateConnectURL\

Description

Sets a boolean in ConnectionFactory indicating whether Connect URL can be updated or not

Function Signature

```
jint acfmetadata_updateConnectURL(FHandle cfmd, jboolean flag);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`flag` - Indicates whether connect URL can be updated or not.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_useThreadContextClassLoader

Description

Sets the value of USE_THREAD_CONTEXT_CLASS_LOADER

Function Signature

```
jint acfmetadata_useThreadContextClassLoaderFHandle cfmd, jboolean
isThreadContextClassLoaderUsed);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isThreadContextClassLoaderUsed` - TRUE or FALSE.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_validateMetaData

Description

Validates if the URL and the name have been specified properly or not.

Function Signature

```
jint acfmetadata_validateMetaData(FHandle cfmd);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_validateURL

Description

Validates the URL to see if the URL is a IP address/hostname It replaces "local-host" with complete IP address.

Function Signature

```
jint acfmetadata_validateURL(FHandle cfmd);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getDescription

Description

Gets the description of the Administered Object The object implements the read-External method to restore its contents by calling the methods of DataInput for primitive types and readObject for objects, strings and arrays.

Function Signature

```
jint acfmetadata_getDescription(FHandle jmd, const char *descrip-
tion);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`description` - A description of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_getName

Description

Gets the Name of the Administered Object.

Function Signature

```
jint acfmetadata_getName(FHandle jmd, const char *name);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`name` - The name of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_hashCode

Description

Returns a hash code value for the object.

Function Signature

```
jint acfmetadata_hashCode(FHandle jmd, jint hc);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`hc` - The has code value of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setDescription

Description

Sets the description of the MetaData Object

Function Signature

```
jint acfmetadata_setDescription(FHandle jmd, const char strDescrip-
tion);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`strDescription` - A description for this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acfmetadata_setName

Description

Sets the Name of this MetaData Object

Function Signature

```
jint acfmetadata_setName(FHandle jmd, const char strName);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`strName` - The name of this meta data object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

QueueConnectionFactoryMetaData

This class represents the MetaData information for a QueueConnectionFactory at it is stored in an LDAP directory. Objects of this class are used to create QueueConnectionFactory objects.

createQueueConnectionFactoryMetaData

Description

Creates a new QueueConnectionFactoryMetaData object.

Function Signature

```
jint createQueueConnectionFactoryMetaData(FHandleqcfmd);
```

Parameters

`qcfmd` - A newly created QueueConnectionFactoryMetaData object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_equals

Description

Compares two QueueConnectionFactoryMetaData objects for equality.

Function Signature

```
jint qcfmetadata_equals(FHandle qcfmd, FHandle obj, jboolean
result);
```

Parameters

`qcfmd` - The QueueConnectionFactoryMetaData object to operate on.

`result` - Will contain TRUE if the two objects are equal, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_allowAutoRevalidation

Description

Sets the value for auto-revalidation.

Function Signature

```
jint qcfmetadata_allowAutoRevalidation(FHandle cfmd, const char
allowAutoRevalidation);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`allowAutoRevalidationtrue` - If auto-revalidation is required, false otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_allowDurableConnections

Description

Enablesthe durable connections for this ConnectionFactory.

Function Signature

```
jint qcfmetadata_allowDurableConnections(FHandle cfmd, jboolean
allowDurableConnections);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`allowDurableConnections` - TRUE if durable connections have to be enabled and FALSE otherwise

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_areDurableConnectionsAllowed

Description

Gives the status whether durable connection are allowed or not for this connection factory.

Function Signature

```
jint qcfmetadata_areDurableConnectionsAllowed(FHandle cfmd, jbool-
ean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if durable connections are allowed, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_disableCSPStoredMessageSend

Description

Disables the sending of any pending messages in the client side store of all durable connections created using this ConnectionFactory.

Function Signature

```
jint qcfmetadata_disableCSPStoredMessageSend(FHandle cfmd, const
char dontSend);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`dontSend` - True if send stored messages has to be disabled else false

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_disablePing

Description

Disables the ping functionality for this ConnectionFactory.

Function Signature

```
jint qcfmetadata_disablePing(FHandle cfmd, jboolean ping);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`ping` - TRUE will disabel pinging, FALSE will disable it.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getAdminConnectionReconnectInterval

Description

Returns the value for the reconnect interval between two reconnect attempts made all durable connections created using this Connection Factory.

Function Signature

```
jint qcfmetadata_getAdminConnectionReconnectInterval(FHandle cfmd,
jlong interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - Will contain the reconnect interval after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getAutoDispatch

Description

Returns the value of the auto dispatch boolean for this Connection Factory.

Function Signature

```
jint qcfmetadata_getAutoDispatch(FHandle cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the value of the auto dispatch boolean as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getBackupConnectURL

Description

Returns the back up connect URL at position num.

Function Signature

```
jint qcfmetadata_getBackupConnectURL(FHandle cfmd, jint num, const
char *url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the backup URL.

`url` - Will contain backup connection URL for this Connection Factory at position num.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getBackupInetAddress

Description

Gets the InetAddress at position num of the BackURL list.

Function Signature

```
jint qcfmetadata_getBackupInetAddress(FHandle cfmd, jint num, FHan-
dle inetAddress);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the InetAddress.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getBackupPort

Description

Gets the port for URL at position num of the BackURL list.

Function Signature

```
jint qcfmetadata_getBackupPort(FHandle cfmd, jint num, jint backup-
Port);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the URL.

`backupPort` - Will contain the backup port number after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getBackupURLStrings

Description

Gets the semicolon seperated string of Backup URLs.

Function Signature

```
jint qcfmetadata_getBackupURLStrings(FHandle cfmd, const char *url-
String);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`urlString -` Will contain the semicolon seperated list of backup URLs.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getBatchTimeoutInterval

Description

Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint qcfmetadata_getBatchTimeoutInterval(FHandle cfmd, const char
*interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - Will contain the CSP syncing delay.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getCBREnabled

Description

Returns true if Content Based Routing is enabled for this Connection Factory and false otherwise.

Function Signature

```
jint qcfmetadata_getCBREnabled(FHandle cfmd, const char *cbrEn-
abled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`cbrEnabled` - Will contain true if CBR is enabled else false.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getClientProxyURL

Description

Returns Client Proxy URL being used for TCP based connections created using this ConnectionFactory.

Function Signature

```
jint qcfmetadata_getClientProxyURL(FHandle cfmd, const char *url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`url` - Will contain the client proxy URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getCompressionManager

Description

Gets the CompressionManager class name to be used for MessageCompression purposes.

Function Signature

```
jint qcfmetadata_getCompressionManager(FHandle cfmd, const char
*compressionManager);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`compressionManager -` Will contain the name of the compression manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getConnectionClientID

Description

Returns the ClientID for this MetaData

Function Signature

```
jint qcfmetadata_getConnectionClientID(FHandle cfmd, const char
*clientID);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`@para clientID` - Will contain the client ID.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getConnectURL

Description

Returns the URL of the FMQ server to which all connections created using this ConnectionFactory will be made.

Function Signature

```
jint qcfmetadata_getConnectURL(FHandle cfmd, const char *url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`url` - Will contain the URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getCreateLocalSocket

Description

Gets the value of the local socket boolean.

Function Signature

```
jint qcfmetadata_getCreateLocalSocket(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain true or false as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getCSPUpdateFrequency

Description

Gets the UpdateFrequency set for this connection factory.

Function Signature

```
jint qcfmetadata_getCSPUpdateFrequency(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the update frequency as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getDurableConnectionBaseDir

Description

Gets the Durable Connection base directory for this ConnectionFactory as a string.

Function Signature

```
jint qcfmetadata_getDurableConnectionBaseDir(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Path of the base directory that is being used for client side storage of messages.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getDurableConnectionReconnectInterval

Description

Returns the value for the reconnect interval between two reconnect attempts made all durable connections created using this Connection Factory.

Function Signature

```
jint qcfmetadata_getDurableConnectionReconnectInterval(FHandle
cfmd, jlong interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - Will contain the reconnect interval.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getFactoryMetadataParams

Description

Returns a Hashtable of all the properties for this ConnectionFactory.

Function Signature

```
jint qcfmetadata_getFactoryMetadataParams(FHandle cfmd, FHandle
ht);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`ht` - Will contain a hashtable of all the properties for the connection factory.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getHTTPProxyURL

Description

Returns HTTP Proxy URL being used for HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint qcfmetadata_getHTTPProxyURL(FHandle cfmd, const char *url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`Url` - Will contain the URL of the HTTP proxy as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getInetAddress

Description

Gets the Inet Address of the Server on which the ConnectionFactory resides in the Server.

Function Signature

```
jint qcfmetadata_getInetAddress(FHandle cfmd, FHandle inetAddress);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`inetAddress` - Will contain the Inet address of the server.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getLMSEnabled

Description

Returns whether support for large messages is enabled or not

Function Signature

```
jint qcfmetadata_getLMSEnabled(FHandle cfmd, jboolean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if large message support is enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getLookUpPreferredServer

Description

Gets the value for the lookup preferred server boolean.

Function Signature

```
jint qcfmetadata_getLookUpPreferredServer(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain true or false as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getMaxAdminConnectionReconnectAttempts

Description

Returns the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server) that are made by a admin connection created using this Connection Factory, as string.

Function Signature

```
jint qcfmetadata_getMaxAdminConnectionReconnectAttempts(FHandle
cfmd, jint result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the maximum number of reconnect attempts.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getMaxDurableConnectionReconnectAttempts

Description

Returns the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server) that are made by a durable connection created using this Connection Factory, as string.

Function Signature

```
jint qcfmetadata_getMaxDurableConnectionReconnectAttempts(FHandle
cfmd, jint result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the maximum number of reconnect attempts.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getMaxSocketCreationTries

Description

Returns the value for the Maximum socket creation tries for all connections made using this ConnectionFactory.

Function Signature

```
jint qcfmetadata_getMaxSocketCreationTries(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the maximum number of socket creation tries as string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getPort

Description

Gets the port of the Server on which the ConnectionFactory resides in the Server.

Function Signature

```
jint qcfmetadata_getPort(FHandle cfmd, jint result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the port number.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getProxyAuthenticationRealm

Description

Returns the Proxy Authentication realm being used on the proxy for HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint qcfmetadata_getProxyAuthenticationRealm(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain name of the proxy authentication realm being used, as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getProxyCredentials

Description

Returns the password for the proxy through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint qcfmetadata_getProxyCredentials(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the password of the proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getProxyPrincipal

Description

Returns the username for the proxy through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint qcfmetadata_getProxyPrincipal(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the username of the proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getProxyType

Description

Returns the proxy name through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint qcfmetadata_getProxyType(FHandle cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the name of the proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getPublishBehaviourInAutoRevalidation

Description

Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint qcfmetadata_getPublishBehaviourInAutoRevalidation(FHandle
cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the CSP sync-up delay as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getPublishWaitDuringCSPSyncp

Description

Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint qcfmetadata_getPublishWaitDuringCSPSyncp(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the CSP sync-up delay.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getSecurityManager

Description

Returns the SecurityManager for this ConnectionFactory. Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint qcfmetadata_getSecurityManager(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the name of the security manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getSecurityProtocol

Description

Returns the security protocol for this ConnectionFactory.

Function Signature

```
jint qcfmetadata_getSecurityProtocol(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - will contain the security protocol.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getServerProxyURL

Description

Returns Server Proxy URL being used for TCP based connections created using this ConnectionFactory.

Function Signature

```
jint qcfmetadata_getServerProxyURL(FHandle cfmd, const
char*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the URL of the server proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getShutdownHookEnabled

Description

Returns whether shutdown hook is enabled or not.

Function Signature

```
jint qcfmetadata_getShutdownHookEnabled(FHandle cfmd, jboolean
result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if shutdown hook is enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getSleepSocketCreationTries

Description

Returns the value for the sleep time between two socket creation tries for all connections] made using this ConnectionFactory.

Function Signature

```
jint qcfmetadata_getSleepSocketCreationTries(FHandle cfmd, const
char sleepTime);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the sleep time bewteen socket creation tries as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getSocketTimeout

Description

Returns the timeout value for the sockets of all connections created using this ConnectionFactory.

Function Signature

```
jint qcfmetadata_getSocketTimeout(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the timeout value as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getSOCKSProxyURL

Description

Returns SOCKS Proxy URL being used for HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint qcfmetadata_getSOCKSProxyURL(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the URI of the SOCKS proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getTCPBatchSize

Description

Gets the message batch size for this ConnectionFactory.

Function Signature

```
jint qcfmetadata_getTCPBatchSize(FHandle cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the batch size as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getTransportProtocol

Description

Returns the Transport Protocol for this ConnectionFactory.

Function Signature

```
jint qcfmetadata_getTransportProtocolFHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the transport protocol being used as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getXASocketTimeout

Description

Returns the timeout value for the socket used for the XAResource prepare and commit calls.

Function Signature

```
jint qcfmetadata_getXASocketTimeout(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the timeout value as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_isAutoRevalidationEnabled

Description

Returns whether auto-revalidation is enabled or not.

Function Signature

```
jint qcfmetadata_isAutoRevalidationEnabled(FHandle cfmd, const
char*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain true if auto-revalidation is enabled, false otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_isConnectURLUpdationAllowed

Description

Indicates whether Connect URL can be updated or not.

Function Signature

```
jint qcfmetadata_isConnectURLUpdationAllowed(FHandle cfmd, jboolean
result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain a boolean indicating whether connect URL can be updated or not.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_isCSPStoredMessageSendDisabled

Description

Checks if send of pending messages is disabled for this connection factory.

Function Signature

```
jint qcfmetadata_isCSPStoredMessageSendDisabled(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - True if send pending is disabled else false; null is returned if no value was set.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_isPingDisabled

Description

Checks if ping is enabled or disabled for this ConnectionFactory.

Function Signature

```
jint qcfmetadata_isPingDisabled(FHandle cfmd, jboolean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if ping was disabled while creating the connection factory and FALSE if otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_isThreadContextClassLoaderUsed

Description

Gives value of USE_THREAD_CONTEXT_CLASS_LOADER.

Function Signature

```
jint qcfmetadata_isThreadContextClassLoaderUsed(FHandle cfmd,
jboolean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain boolean specifying whether USE_THREAD_CONTEXT_CLASS_LOADER is true or false.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_loadEnvParams

Description

Loads the properties from the Env that are not already specified in the metaData.

Function Signature

```
jint qcfmetadata_loadEnvParams(FHandle cfmd, FHandle env);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`env` - The Env properties that need to be loaded into the metaData.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setAdminConnectionReconnectInterval

Description

Sets the value for the reconnect interval which is the interval between two successive reconnect attempts that would be made by a Durable connection created using this Connection Factory.

Function Signature

```
jint qcfmetadata_setAdminConnectionReconnectInterval(FHandle cfmd,
const char interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - The reconnect interval as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setAutoDispatch

Description

Sets the value for the auto dispatch boolean.

Function Signature

```
jint qcfmetadata_setAutoDispatch(FHandle cfmd, const char autoDis-
patch);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`autoDispatch` - True or false passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setBackupConnectURL

Description

Sets the back up connect url at position num for this Connection Factory.

Function Signature

```
jint qcfmetadata_setBackupConnectURL(FHandle cfmd, jint num, const
char strBackupConnectURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the backup URL (0 for first backup URL).

`strBackupURL` - The backup URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setBackupConnectURLs

Description

Sets the Backup URL's.

Function Signature

```
jint qcfmetadata_setBackupConnectURLs(FHandle cfmd, const char
backupUrls);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`backupUrls` - A semicolon seperated list of URLs.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setBatchTimeoutInterval

Description

Sets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint qcfmetadata_setBatchTimeoutInterval(FHandle cfmd, const char
batchTimeoutInterval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`batchTimeoutInterval` - The delay in publishing new messages when CSP is syncing up.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setCBREnabled

Description

Enable Content Based Routing for all connections that will be created using this ConnectionFactory.

Function Signature

```
jint qcfmetadata_setCBREnabled(FHandle cfmd, const char isCBREn-
abled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isCBREnabled` - True to enable Content Based Routing else false, passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setClientProxyURL

Description

Sets the URL for the Client side Proxy through which TCP based Connections created using this ConnectionFactory will be routed using HTTP Tunneling.

Function Signature

```
jint qcfmetadata_setClientProxyURL(FHandle cfmd, const char prox-
yURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The client proxy URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setCompressionManager

Description

Sets the CompressionManager class name to be used for MessageCompresson purposes.

Function Signature

```
jint qcfmetadata_setCompressionManager(FHandle cfmd, const char
nmanager);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`manager` - Name of the compression manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setConnectionClientID

Description

Sets the ClientID for this MetaData.

Function Signature

```
jint qcfmetadata_setConnectionClientID(FHandle cfmd, const char
clientID);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`clientID` - The client ID.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setConnectURL

Description

Sets the primary Connect URL for this ConnectionFactory.

Function Signature

```
jint qcfmetadata_setConnectURL(FHandle cfmd, const char strConnec-
tURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setCreateLocalSocket

Description

Sets the value for the local socket boolean.

Function Signature

```
jint qcfmetadata_setCreateLocalSocket(FHandle cfmd, const char
localSocket);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`localSocket` - True of false as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setCSPUpdateFrequency

Description

Sets the UpdateFrequency to the value passed.

Function Signature

```
jint qcfmetadata_setCSPUpdateFrequency(FHandle cfmd, const char
frequency);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`frequency` - The CSP update frequency.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setDurableConnectionBaseDir

Description

Sets the Durable Connection Base directory for this ConnectionFactory as a string.

Function Signature

```
jint qcfmetadata_setDurableConnectionBaseDir(FHandle cfmd, const
char durBaseDir);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`durBaseDir` - Fully qualified path of the base directory for client side storage of messages

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setDurableConnectionReconnectInterval

Description

Sets the value for the reconnect interval which is the interval between two successive reconnect attempts that would be made by a Durable connection created using this Connection Factory.

Function Signature

```
jint qcfmetadata_setDurableConnectionReconnectInterval(FHandle
cfmd, const char interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - The reconnect interval passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setFactoryMetadataParams

Description

Sets the Hashtable containing all properties for this Connection Factory.

Function Signature

```
jint qcfmetadata_setFactoryMetadataParams(FHandle cfmd, FHandle
params);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`params` - Hashtable containing the name value pairs of all Connection Factory properties.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setHTTPProxyURL

Description

Sets the URL for the HTTP Proxy through which HTTP based Connections created using this ConnectionFactory will be routed.

Function Signature

```
jint qcfmetadata_setHTTPProxyURL(FHandle cfmd, const char prox-
yURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The URL of the HTTP proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setLMSEnabled

Description

Enables/disables large message support.

Function Signature

```
jint qcfmetadata_setLMSEnabled(FHandle cfmd, jboolean isLMSEn-
abled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isLMSEnabled` - TRUE if Large support has to be enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setLookUpPreferredServer

Description

Sets the value for the lookup preferred server boolean.

Function Signature

```
jint qcfmetadata_setLookUpPreferredServer(FHandle cfmd, const char
lookupPreferred);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`lookupPreferred` - True or false passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setMaxAdminConnectionReconnectAttempts

Description

Sets the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server) that will be made by an admin connection created using this Connection Factory.

Function Signature

```
jint qcfmetadata_setMaxAdminConnectionReconnectAttempts(FHandle
cfmd,const char numAttempts);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`numAttempts` - Maximum number of reconnect attempts passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setMaxDurableConnectionReconnectAttempts

Description

Sets the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server) that will be made by a durable connection created using this Connection Factory.

Function Signature

```
jint qcfmetadata_setMaxDurableConnectionReconnectAttempts(FHandle
cfmd, const char numAttempts);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`numAttempts` - Maximum number of reconnect attempts passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setMaxSocketCreationTries

Description

Sets the value for the maximum socket creation tries that the FioranoMQ runtime will make for creating the socket for all connections made using this ConnectionFactory.

Function Signature

```
jint qcfmetadata_setMaxSocketCreationTries(FHandle cfmd, const char
socketTries);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`socketTries` - The number of socket creation tries passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setProxyAuthenticationRealm

Description

Sets the Authentication Realm being used on the Proxy through which HTTP based Connections created using this ConnectionFactory will be routed.

Function Signature

```
jint qcfmetadata_setProxyAuthenticationRealm(FHandle cfmd, const
char realm);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`realm` - Name of the proxy authentication realm to be used.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setProxyCredentials

Description

Sets the password for getting validated at the proxy (if required) which is to be used for the HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint qcfmetadata_setProxyCredentials(FHandle cfmd, const char pass-
word);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`password` - The password.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setProxyPrincipal

Description

Sets the username for the proxy (if required) through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint qcfmetadata_setProxyPrincipal(FHandle cfmd, const char username);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`username` - The username.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setProxyType

Description

Sets the type (name) of the proxy being used for the HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint qcfmetadata_setProxyType(FHandle cfmd, const char proxy);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxy` - The type of the proxy as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setPublishBehaviourInAutoRevalidation

Description

Sets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint qcfmetadata_setPublishBehaviourInAutoRevalidation(FHandle
cfmd, const char publishBehaviour);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`publishBehaviour` - Publisher behaviour on auto-revalidation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setPublishWaitDuringCSPSyncp

Description

Sets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint qcfmetadata_setPublishWaitDuringCSPSyncp(FHandle cfmd, const
char delay);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`delay` - The delay passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setSecurityManager

Description

Sets the security manager that will be used to make SSL connections using this ConnectionFactory.

Function Signature

```
jint qcfmetadata_setSecurityManager(FHandle cfmd, const charFunc-
tion Signaturemanager);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`manager` - The name of the security manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setSecurityProtocol

Description

Sets the security protocol over which all connections created using this ConnectionFactory communicate with the FMQ server

Function Signature

```
jint qcfmetadata_setSecurityProtocol(FHandle cfmd, const char protocol);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`protocol` - Security protocol, possible values are PHAOS_SSL and JSSE_SSL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setServerProxyURL

Description

Sets the URL for the Server side Proxy through which TCP based Connections created using this ConnectionFactory will be routed using HTTP Tunneling.

Function Signature

```
jint qcfmetadata_setServerProxyURL(FHandle cfmd, const char proxyURL);)
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The URL of the server proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setShutdownHookEnabled

Description

Enables/disables shutdown hook.

Function Signature

```
jint qcfmetadata_setShutdownHookEnabled(FHandle cfmd, jboolean
isShutdownHookEnabled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isShutdownHookEnabled` - TRUE if shutdown hook had to be enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setSleepSocketCreationTries

Description

Sets the value for the sleep time bewteen two creation tries that the FioranoMQ runtime will make for creating the socket for all connections made using this ConnectionFactory.

Function Signature

```
jint qcfmetadata_setSleepSocketCreationTries(FHandle cfmd, const
char sleepTime);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setSocketTimeout

Description

Sets the timeout value for the sockets of all connections created using this ConnectionFactory.

Function Signature

```
jint qcfmetadata_setSocketTimeout(FHandle cfmd, const char time-
out);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`timeout` - The timeout value passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setSOCKSProxyURL

Description

Sets the URL for the SOCKS Proxy through which HTTP based Connections created using this ConnectionFactory will be routed.

Function Signature

```
jint qcfmetadata_setSOCKSProxyURL(FHandle cfmd, const char prox-
yURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The URL of the SOCKS proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setTCPBatchSize

Description

Sets the value of the message batch size that will be used for NP messages being sent using all connections created using this Connection Factory.

Function Signature

```
jint qcfmetadata_setTCPBatchSize(FHandle cfmd, const char batch-
Size);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`batchSize` - The message batch size that will be used for NP messages.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setTransportProtocol

Description

Sets the transport protocol over which all connections created using this ConnectionFactory communicate with the FMQ server.

Function Signature

```
jint qcfmetadata_setTransportProtocol(FHandle cfmd, const char protocol);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`protocol` - The protocol. Possible values are TCP or HTTP.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setXASocketTimeout

Description

Sets the timeout value for the socket when the XAResource prepare or commit call is made from the runtime.

Function Signature

```
jint qcfmetadata_setXASocketTimeout(FHandle cfmd, const char timeout);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`timeout` - The timeout value as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_updateConnectURL

Description

Sets a boolean in ConnectionFactory indicating whether Connect URL can be updated or not

Function Signature

```
jint qcfmetadata_updateConnectURL(FHandle cfmd, jboolean flag);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`flag` - Indicates whether connect URL can be updated or not.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_useThreadContextClassLoader

Description

Sets the value of USE_THREAD_CONTEXT_CLASS_LOADER

Function Signature

```
jint qcfmetadata_useThreadContextClassLoader(FHandle cfmd, jboolean
isThreadContextClassLoaderUsed);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isThreadContextClassLoaderUsed` - TRUE or FALSE.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_validateMetaData

Description

Validates if the URL and the name have been specified properly or not.

Function Signature

```
jint qcfmetadata_validateMetaData(FHandle cfmd);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_validateURL

Description

Validates the URL to see if the URL is a IP address/hostname It replaces "local-host" with complete IP address.

Function Signature

```
jint qcfmetadata_validateURL(FHandle cfmd);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getDescription

Description

Gets the description of the Administered Object The object implements the read-External method to restore its contents by calling the methods of DataInput for primitive types and readObject for objects, strings and arrays.

Function Signature

```
jint qcfmetadata_getDescription(FHandle jmd, const char **descrip-
tion);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`description` - A description of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_getName

Description

Gets the Name of the Administered Object.

Function Signature

```
jint qcfmetadata_getName(FHandle jmd, const char *name);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`name` - The name of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_hashCode

Description

Returns a hash code value for the object.

Function Signature

```
jint qcfmetadata_hashCode(FHandle jmd, const char *name);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`hc` - The has code value of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setDescription

Description

Sets the description of the MetaData Object

Function Signature

```
jint qcfmetadata_setDescription(FHandle jmd, const char strDescrip-
tion)
```

Parameters

`jmd` - The JMS meta data object to operate on.

`strDescription` - A description for this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcfmetadata_setName

Description

Sets the Name of this MetaData Object

Function Signature

```
jint qcfmetadata_setName(FHandle jmd, const char strName);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`strName` - The name of this meta data object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

TopicConnectionFactoryMetaData

This class represents the MetaData information for a TopicConnectionFactory at it is stored in an LDAP directory. Objects of this class are used to create TopicConnectionFactory objects.

createTopicConnectionFactoryMetaData

Description

Creates a new TopicConnectionFactoryMetaData object.

Function Signature

```
jint createTopicConnectionFactoryMetaData(FHandleFunction Signa-
turetcfmd);
```

Parameters

`tcfmd` - A newly created TopicConnectionFactoryMetaData object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_equals

Description

Compares two TopicConnectionFactoryMetaData objects for equality.

Function Signature

```
jint tcfmetadata_equals(FHandle tcfmd, FHandle obj, jbooleanFunc-
tion Signatureresult);
```

Parameters

`tcfmd` - The TopicConnectionFactoryMetaData object to operate on.

`result` - Will contain TRUE if the two objects are equal, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_allowAutoRevalidation

Description

Sets the value for auto-revalidation.

Function Signature

```
jint tcfmetadata_allowAutoRevalidation(FHandle jmd, const char
*description);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`allowAutoRevalidationtrue` - If auto-revalidation is required, false otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_allowDurableConnections

Description

Enablesthe durable connections for this ConnectionFactory.

Function Signature

```
jint tcfmetadata_allowDurableConnections(FHandle cfmd, jboolean
allowDurableConnections);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`allowDurableConnections` - TRUE if durable connections have to be enabled and FALSE otherwise

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_areDurableConnectionsAllowed

Description

Gives the status whether durable connection are allowed or not for this connection factory.

Function Signature

```
jint tcfmetadata_areDurableConnectionsAllowed(FHandle jmd, const
char *name);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if durable connections are allowed, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_disableCSPStoredMessageSend

Description

Disables the sending of any pending messages in the client side store of all durable connections created using this ConnectionFactory.

Function Signature

```
jint tcfmetadata_disableCSPStoredMessageSend(FHandle cfmd, const
charFunction Signature dontSend);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`dontSend` - True if send stored messages has to be disabled else false

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_disablePing

Description

Disables the ping functionality for this ConnectionFactory.

Function Signature

```
jint tcfmetadata_disablePing(FHandle cfmd, jboolean ping);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`ping` - TRUE will disabel pinging, FALSE will disable it.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getAdminConnectionReconnectInterval

Description

Returns the value for the reconnect interval between two reconnect attempts made all durable connections created using this Connection Factory.

Function Signature

```
jint tcfmetadata_getAdminConnectionReconnectInterval(FHandle cfmd,
jlong interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - Will contain the reconnect interval after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getAutoDispatch

Description

Returns the value of the auto dispatch boolean for this Connection Factory.

Function Signature

```
jint tcfmetadata_getAutoDispatch(FHandle cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the value of the auto dispatch boolean as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getBackupConnectURL

Description

Returns the back up connect URL at position num.

Function Signature

```
jint tcfmetadata_getBackupConnectURL(FHandle cfmd, jint num, const
char *url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the backup URL.

`url` - Will contain backup connection URL for this Connection Factory at position num.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getBackupInetAddress

Description

Gets the InetAddress at position num of the BackURL list.

Function Signature

```
jint tcfmetadata_getBackupInetAddress(FHandle cfmd, jint num, FHandle inetAddress);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the InetAddress.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getBackupPort

Description

Gets the port for URL at position num of the BackURL list.

Function Signature

```
jint tcfmetadata_getBackupPort(FHandle cfmd, jint num, jint backupPort);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the URL.

`backupPort` - Will contain the backup port number after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getBackupURLStrings

Description

Gets the semicolon seperated string of Backup URLs.

Function Signature

```
jint tcfmetadata_getBackupURLStrings(FHandle cfmd, const char *url-
String);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`urlString` - Will contain the semicolon seperated list of backup URLs.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getBatchTimeoutInterval

Description

Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint tcfmetadata_getBatchTimeoutInterval(FHandle cfmd, const char
*interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - Will contain the CSP syncing delay.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getCBREnabled

Description

Returns true if Content Based Routing is enabled for this Connection Factory and false otherwise.

Function Signature

```
jint tcfmetadata_getCBREnabled(FHandle cfmd, const char *cbrEn-
abled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`cbrEnabled` - Will contain true if CBR is enabled else false.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getClientProxyURL

Description

Returns Client Proxy URL being used for TCP based connections created using this ConnectionFactory.

Function Signature

```
jint tcfmetadata_getClientProxyURL(FHandle cfmd, const char *url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`url` - Will contain the client proxy URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getCompressionManager

Description

Gets the CompressionManager class name to be used for MessageCompression purposes.

Function Signature

```
jint tcfmetadata_getCompressionManager(FHandle cfmd, const char
*compressionManager);
```

Parameters

`cfmd` the connection factory meta data object to operate on.

`compressionManager` will contain the name of the compression manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getConnectionClientID

Description

Returns the ClientID for this MetaData

Function Signature

```
jint tcfmetadata_getConnectionClientID(FHandle cfmd, const char
*clientID);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`@para clientID` - Will contain the client ID.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getConnectURL

Description

Returns the URL of the FMQ server to which all connections created using this ConnectionFactory will be made.

Function Signature

```
jint tcfmetadata_getConnectURL(FHandle cfmd, const char *url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`url` - Will contain the URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getCreateLocalSocket

Description

Gets the value of the local socket boolean.

Function Signature

```
jint tcfmetadata_getCreateLocalSocket(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain true or false as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getCSPUpdateFrequency

Description

Gets the UpdateFrequency set for this connection factory.

Function Signature

```
jint tcfmetadata_getCSPUpdateFrequency(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the update frequency as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getDurableConnectionBaseDir

Description

Gets the Durable Connection base directory for this ConnectionFactory as a string.

Function Signature

```
jint tcfmetadata_getDurableConnectionBaseDir(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Path of the base directory that is being used for client side storage of messages.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getDurableConnectionReconnectInterval

Description

Returnsthe value for the reconnect interval between two reconnect attempts made all durable connections created using this Connection Factory.

Function Signature

```
jint tcfmetadata_getDurableConnectionReconnectInterval(FHandle
cfmd, jlong interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - Will contain the reconnect interval.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getFactoryMetadataParams

Description

Returnsa Hashtable of all the properties for this ConnectionFactory.

Function Signature

```
jint tcfmetadata_getFactoryMetadataParams(FHandle cfmd, FHandle
ht);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`ht` - Will contain a hashtable of all the properties for the connection factory.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getHTTPProxyURL

Description

Returns HTTP Proxy URL being used for HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint tcfmetadata_getHTTPProxyURL(FHandle cfmd, const char *url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`url` - Will contain the URL of the HTTP proxy as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getInetAddress

Description

Gets the Inet Address of the Server on which the ConnectionFactory resides in the Server.

Function Signature

```
jint tcfmetadata_getInetAddress(FHandle cfmd, FHandle inetAddress);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`inetAddress` - Will contain the Inet address of the server.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getLMSEnabled

Description

Returns whether support for large messages is enabled or not

Function Signature

```
jint tcfmetadata_getLMSEnabled(FHandle cfmd, jboolean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if large message support is enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getLookUpPreferredServer

Description

Gets the value for the lookup preferred server boolean.

Function Signature

```
jint tcfmetadata_getLookUpPreferredServer(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain true or false as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getMaxAdminConnectionReconnectAttempts

Description

Returns the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server) that are made by a admin connection created using this Connection Factory, as string.

Function Signature

```
jint tcfmetadata_getMaxAdminConnectionReconnectAttempts(FHandle
cfmd, jint result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the maximum number of reconnect attempts.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getMaxDurableConnectionReconnectAttempts

Description

Returns the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server) that are made by a durable connection created using this Connection Factory, as string.

Function Signature

```
jint tcfmetadata_getMaxDurableConnectionReconnectAttempts(FHandle
cfmd, jint result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the maximum number of reconnect attempts.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getMaxSocketCreationTries

Description

Returns the value for the Maximum socket creation tries for all connections] made using this ConnectionFactory.

Function Signature

```
jint tcfmetadata_getMaxSocketCreationTries(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - the connection factory meta data object to operate on.

`result` - Will contain the maximum number of socket creation tries as string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getPort

Description

Gets the port of the Server on which the ConnectionFactory resides in the Server.

Function Signature

```
jint tcfmetadata_getPort(FHandle cfmd, jint result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the port number.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getProxyAuthenticationRealm

Description

Returns the Proxy Authentication realm being used on the proxy for HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint tcfmetadata_getProxyAuthenticationRealm(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain name of the proxy authentication realm being used, as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getProxyCredentials

Description

Returns the password for the proxy through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint tcfmetadata_getProxyCredentials(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the password of the proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getProxyPrincipal

Description

Returns the username for the proxy through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint tcfmetadata_getProxyPrincipal(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the username of the proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getProxyType

Description

Returns the proxy name through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint tcfmetadata_getProxyType(FHandle cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the name of the proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getPublishBehaviourInAutoRevalidation

Description

Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint tcfmetadata_getPublishBehaviourInAutoRevalidation(FHandle
cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the CSP sync-up delay as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getPublishWaitDuringCSPSyncp

Description

Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint tcfmetadata_getPublishWaitDuringCSPSyncp(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the CSP sync-up delay.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getSecurityManager

Description

Returns the SecurityManager for this ConnectionFactory. Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint tcfmetadata_getSecurityManager (FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the name of the security manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getSecurityProtocol

Description

Returns the security protocol for this ConnectionFactory.

Function Signature

```
jint tcfmetadata_getSecurityProtocol(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the security protocol.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getServerProxyURL

Description

Returns Server Proxy URL being used for TCP based connections created using this ConnectionFactory.

Function Signature

```
jint tcfmetadata_getServerProxyURL(FHandle cfmd, const
char*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the URL of the server proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getShutdownHookEnabled

Description

Returns whether shutdown hook is enabled or not.

Function Signature

```
jint tcfmetadata_getShutdownHookEnabled(FHandle cfmd, jboolean
result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if shutdown hook is enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getSleepSocketCreationTries

Description

Returns the value for the sleep time between two socket creation tries for all connections made using this ConnectionFactory.

Function Signature

```
jint tcfmetadata_getSleepSocketCreationTries(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the sleep time bewteen socket creation tries as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getSocketTimeout

Description

Returns the timeout value for the sockets of all connections created using this ConnectionFactory.

Function Signature

```
jint tcfmetadata_getSocketTimeout(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the timeout value as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getSOCKSProxyURL

Description

Returns SOCKS Proxy URL being used for HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint tcfmetadata_getSOCKSProxyURL(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the URI of the SOCKS proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getTCPBatchSize

Description

Gets the message batch size for this ConnectionFactory.

Function Signature

```
jint tcfmetadata_getTCPBatchSize(FHandle cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the batch size as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getTransportProtocol

Description

Returns the Transport Protocol for this ConnectionFactory.

Function Signature

```
jint tcfmetadata_getTransportProtocol(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the transport protocol being used as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getXASocketTimeout

Description

Returns the timeout value for the socket used for the XAResource prepare and commit calls.

Function Signature

```
jint tcfmetadata_getXASocketTimeout(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the timeout value as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_isAutoRevalidationEnabled

Description

Returns whether auto-revalidation is enabled or not.

Function Signature

```
jint tcfmetadata_isAutoRevalidationEnabled(FHandle cfmd, const
char*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain true if auto-revalidation is enabled, false otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_isConnectURLUpdationAllowed

Description

Indicates whether Connect URL can be updated or not.

Function Signature

```
jint tcfmetadata_isConnectURLUpdationAllowed(FHandle cfmd, jboolean
result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain a boolean indicating whether connect URL can be updated or not.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_isCSPStoredMessageSendDisabled

Description

Checks if send of pending messages is disabled for this connection factory.

Function Signature

```
jint tcfmetadata_isCSPStoredMessageSendDisabled(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - True if send pending is disabled else false; null is returned if no value was set.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_isPingDisabled

Description

Checks if ping is enabled or disabled for this ConnectionFactory.

Function Signature

```
jint tcfmetadata_isPingDisabled(FHandle cfmd, jboolean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if ping was disabled while creating the connection factory and FALSE if otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_isThreadContextClassLoaderUsed

Description

Gives value of USE_THREAD_CONTEXT_CLASS_LOADER.

Function Signature

```
jint tcfmetadata_isThreadContextClassLoaderUsed(FHandle cfmd,
jboolean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain boolean specifying whether USE_THREAD_CONTEXT_CLASS_LOADER is true or false.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_loadEnvParams

Description

Loads the properties from the Env that are not already specified in the metaData.

Function Signature

```
jint tcfmetadata_loadEnvParams(FHandle cfmd, FHandle env);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`env` - The Env properties that need to be loaded into the metaData.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setAdminConnectionReconnectInterval

Description

Sets the value for the reconnect interval which is the interval between two successive reconnect attempts that would be made by a Durable connection created using this Connection Factory.

Function Signature

```
jint tcfmetadata_setAdminConnectionReconnectInterval(FHandle cfmd,
const char interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - The reconnect interval as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setAutoDispatch

Description

Sets the value for the auto dispatch boolean.

Function Signature

```
jint tcfmetadata_setAutoDispatch(FHandle cfmd, const char autoDis-
patch);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`autoDispatch` - True or false passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setBackupConnectURL

Description

Sets the back up connect url at position num for this Connection Factory.

Function Signature

```
jint tcfmetadata_setBackupConnectURL(FHandle cfmd, jint num, const
char strBackupConnectURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the backup URL (0 for first backup URL).

`strBackupURL` - The backup URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setBackupConnectURLs

Description

Sets the Backup URL's.

Function Signature

```
jint tcfmetadata_setBackupConnectURLs(FHandle cfmd, const char
backupUrls);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`backupUrls` - A semicolon seperated list of URLs.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setBatchTimeoutInterval

Description

Sets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint tcfmetadata_setBatchTimeoutIntervalFHandle cfmd, const char
batchTimeoutInterval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`batchTimeoutInterval` - The delay in publishing new messages when CSP is syncing up.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setCBREnabled

Description

Enable Content Based Routing for all connections that will be created using this ConnectionFactory.

Function Signature

```
jint tcfmetadata_setCBREnabled(FHandle cfmd, const char isCBREn-
abled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isCBREnabled` - True to enable Content Based Routing else false, passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setClientProxyURL

Description

Sets the URL for the Client side Proxy through which TCP based Connections created using this ConnectionFactory will be routed using HTTP Tunneling.

Function Signature

```
jint tcfmetadata_setClientProxyURL(FHandle cfmd, const char prox-
yURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The client proxy URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setCompressionManager

Description

Sets the CompressionManager class name to be used for MessageCompresson purposes.

Function Signature

```
jint tcfmetadata_setCompressionManager(FHandle cfmd, const char
nmanager);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`manager` - Name of the compression manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setConnectionClientID

Description

Sets the ClientID for this MetaData.

Function Signature

```
jint tcfmetadata_setConnectionClientID(FHandle cfmd, const char
clientID);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`clientID` - The client ID.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setConnectURL

Description

Sets the primary Connect URL for this ConnectionFactory.

Function Signature

```
jint tcfmetadata_setConnectURL(FHandle cfmd, const char strConnec-
tURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setCreateLocalSocket

Description

Sets the value for the local socket boolean.

Function Signature

```
jint tcfmetadata_setCreateLocalSocket(FHandle cfmd, const char
localSocket);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`localSocket` - True of false as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setCSPUpdateFrequency

Description

Sets the UpdateFrequency to the value passed.

Function Signature

```
jint tcfmetadata_setCSPUpdateFrequency(FHandle cfmd, const char
frequency);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`frequency` - The CSP update frequency.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setDurableConnectionBaseDir

Description

Sets the Durable Connection Base directory for this ConnectionFactory as a string.

Function Signature

```
jint tcfmetadata_setDurableConnectionBaseDir(FHandle cfmd, const
char durBaseDir);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`durBaseDir` - Fully qualified path of the base directory for client side storage of messages

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setDurableConnectionReconnectInterval

Description

Sets the value for the reconnect interval which is the interval between two successive reconnect attempts that would be made by a Durable connection created using this Connection Factory.

Function Signature

```
jint tcfmetadata_setDurableConnectionReconnectInterval(FHandle
cfmd, const char interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - The reconnect interval passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setFactoryMetadataParams

Description

Sets the Hashtable containing all properties for this Connection Factory.

Function Signature

```
jint tcfmetadata_setFactoryMetadataParams(FHandle cfmd, FHandle
params);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`params` - Hashtable containing the name value pairs of all Connection Factory properties.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setHTTPProxyURL

Description

Sets the URL for the HTTP Proxy through which HTTP based Connections created using this ConnectionFactory will be routed.

Function Signature

```
jint tcfmetadata_setHTTPProxyURL(FHandle cfmd, const char proxyURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The URL of the HTTP proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setLMSEnabled

Description

Enables/disables large message support.

Function Signature

```
jint tcfmetadata_setLMSEnabled(FHandle cfmd, jboolean isLMSEnabled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isLMSEnabled` - TRUE if Large support has to be enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setLookUpPreferredServer

Description

Sets the value for the lookup preferred server boolean.

Function Signature

```
jint tcfmetadata_setLookUpPreferredServer(FHandle cfmd, const char
lookupPreferred);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`lookupPreferred` - True or false passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setMaxAdminConnectionReconnectAttempts

Description

Sets the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server) that will be made by an admin connection created using this Connection Factory.

Function Signature

```
jint tcfmetadata_setMaxAdminConnectionReconnectAttempts(FHandle
cfmd,const char numAttempts);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`numAttempts` - Maximum number of reconnect attempts passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setMaxDurableConnectionReconnectAttempts

Description

Sets the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server) that will be made by a durable connection created using this Connection Factory.

Function Signature

```
jint tcfmetadata_setMaxDurableConnectionReconnectAttempts(FHandle cfmd, const char numAttempts);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`numAttempts` - Maximum number of reconnect attempts passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setMaxSocketCreationTries

Description

Sets the value for the maximum socket creation tries that the FioranoMQ runtime will make for creating the socket for all connections made using this Connection-Factory.

Function Signature

```
jint tcfmetadata_setMaxSocketCreationTries(FHandle cfmd, const char socketTries);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`socketTries` - The number of socket creation tries passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setProxyAuthenticationRealm

Description

Sets the Authentication Realm being used on the Proxy through which HTTP based Connections created using this ConnectionFactory will be routed.

Function Signature

```
jint tcfmetadata_setProxyAuthenticationRealm(FHandle cfmd, const char realm);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`realm` - Name of the proxy authentication realm to be used.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setProxyCredentials

Description

Sets the password for getting validated at the proxy (if required) which is to be used for the HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint tcfmetadata_setProxyCredentials(FHandle cfmd, const char password);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`password` - The password.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setProxyPrincipal

Description

Sets the username for the proxy (if required) through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint tcfmetadata_setProxyPrincipal(FHandle cfmd, const char user-
name);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`username` - The username.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setProxyType

Description

Sets the type (name) of the proxy being used for the HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint tcfmetadata_setProxyType(FHandle cfmd, const char proxy);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxy` - The type of the proxy as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setPublishBehaviourInAutoRevalidation

Description

Sets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint tcfmetadata_setPublishBehaviourInAutoRevalidation(FHandle
cfmd, const char publishBehaviour);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`publishBehaviour` - Publisher behaviour on auto-revalidation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setPublishWaitDuringCSPSyncp

Description

Sets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint tcfmetadata_setPublishWaitDuringCSPSyncp(FHandle cfmd, const
char delay);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`delay` - The delay passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setSecurityManager

Description

Sets the security manager that will be used to make SSL connections using this ConnectionFactory.

Function Signature

```
jint tcfmetadata_setSecurityManager(FHandle cfmd, const char man-
ager);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`manager` - The name of the security manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setSecurityProtocol

Description

Sets the security protocol over which all connections created using this ConnectionFactory communicate with the FMQ server

Function Signature

```
jint tcfmetadata_setSecurityProtocol(FHandle cfmd, const char protocol);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`protocol` - Security protocol, possible values are PHAOS_SSL and JSSE_SSL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setServerProxyURL

Description

Sets the URL for the Server side Proxy through which TCP based Connections created using this ConnectionFactory will be routed using HTTP Tunneling.

Function Signature

```
jint tcfmetadata_setServerProxyURL(FHandle cfmd, const char proxyURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The URL of the server proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setShutdownHookEnabled

Description

Enables/disables shutdown hook.

Function Signature

```
jint tcfmetadata_setShutdownHookEnabled(FHandle cfmd, jboolean
isShutdownHookEnabled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isShutdownHookEnabled` - TRUE if shutdown hook had to be enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setSleepSocketCreationTries

Description

Sets the value for the sleep time bewteen two creation tries that the FioranoMQ runtime will make for creating the socket for all connections made using this ConnectionFactory.

Function Signature

```
jint tcfmetadata_setSleepSocketCreationTries(FHandle cfmd, const
char sleepTime);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setSocketTimeout

Description

Sets the timeout value for the sockets of all connections created using this ConnectionFactory.

Function Signature

```
jint tcfmetadata_setSocketTimeout(FHandle cfmd, const char time-
out);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`timeout` - The timeout value passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setSOCKSProxyURL

Description

Sets the URL for the SOCKS Proxy through which HTTP based Connections created using this ConnectionFactory will be routed.

Function Signature

```
jint tcfmetadata_setSOCKSProxyURL(FHandle cfmd, const char proxyURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The URL of the SOCKS proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setTCPBatchSize

Description

Sets the value of the message batch size that will be used for NP messages being sent using all connections created using this Connection Factory.

Function Signature

```
jint tcfmetadata_setTCPBatchSize(FHandle cfmd, const char batchSize);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`batchSize` - The message batch size that will be used for NP messages.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setTransportProtocol

Description

Sets the transport protocol over which all connections created using this ConnectionFactory communicate with the FMQ server.

Function Signature

```
jint tcfmetadata_setTransportProtocol(FHandle cfmd, const char protocol);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`protocol` - The protocol. Possible values are TCP or HTTP.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setXASocketTimeout

Description

Sets the timeout value for the socket when the XAResource prepare or commit call is made from the runtime.

Function Signature

```
jint tcfmetadata_setXASocketTimeout(FHandle cfmd, const char timeout);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`timeout` - The timeout value as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_updateConnectURL

Description

Sets a boolean in ConnectionFactory indicating whether Connect URL can be updated or not

Function Signature

```
jint tcfmetadata_updateConnectURL(FHandle cfmd, jboolean flag);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`flag` - Indicates whether connect URL can be updated or not.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_useThreadContextClassLoader

Description

Sets the value of USE_THREAD_CONTEXT_CLASS_LOADER

Function Signature

```
jint tcfmetadata_useThreadContextClassLoader(FHandle cfmd, jboolean
isThreadContextClassLoaderUsed);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isThreadContextClassLoaderUsed` - TRUE or FALSE.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_validateMetaData

Description

Validate if the URL and the name have been specified properly or not.

Function Signature

```
jint tcfmetadata_validateMetaData(FHandle cfmd);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_validateURL

Description

Validates the URL to see if the URL is a IP address/hostname It replaces "local-host" with complete IP address.

Function Signature

```
jint tcfmetadata_validateURL(FHandle cfmd);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getDescription

Description

Gets the description of the Administered Object The object implements the read-External method to restore its contents by calling the methods of DataInput for primitive types and readObject for objects, strings and arrays.

Function Signature

```
jint tcfmetadata_getDescription(FHandle jmd, const char *descrip-
tion);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`description` - A description of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_getName

Description

Gets the Name of the Administered Object.

Function Signature

```
jint tcfmetadata_getName(FHandle jmd, const char *name);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`name` - The name of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_hashCode

Description

Returns a hash code value for the object.

Function Signature

```
jint tcfmetadata_hashCode(FHandle jmd, jint hc);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`hc` - The has code value of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setDescription

Description

Sets the description of the MetaData Object

Function Signature

```
jint tcfmetadata_setDescription(FHandle jmd, const char strDescrip-
tion);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`strDescription` - A description for this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcfmetadata_setName

Description

Sets the Name of this MetaData Object

Function Signature

```
jint tcfmetadata_setName(FHandle jmd, const char strName);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`strName` - The name of this meta data object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

UnifiedConnectionFactoryMetaData

This class represents the MetaData information for a CommonConnectionFactory at it is stored in an LDAP directory. Objects of this class are used to create CommonConnectionFactory objects.

createUnifiedConnectionFactoryMetaData

Description

Creates a new UnifiedConnectionFactoryMetaData object.

Function Signature

```
jint createUnifiedConnectionFactoryMetaData(FHandle ucfmd);
```

Parameters

`ucfmd` - A newly created UnifiedConnectionFactoryMetaData object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_equals

Description

Compares two UnifiedConnectionFactoryMetaData objects for equality.

Function Signature

```
jint ucfmetadata_equals(FHandle ucfmd, FHandle obj, jboolean
result);
```

Parameters

`ucfmd` - The UnifiedConnectionFactoryMetaData object to operate on.

`result` - Will contain TRUE if the two objects are equal, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_allowAutoRevalidation

Description

Sets the value for auto-revalidation.

Function Signature

```
jint ucfmetadata_allowAutoRevalidation(FHandle cfmd, const char
allowAutoRevalidation);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`allowAutoRevalidationtrue` - If auto-revalidation is required, false otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_allowDurableConnections

Description

Enablesthe durable connections for this ConnectionFactory.

Function Signature

```
jint ucfmetadata_allowDurableConnections(FHandle cfmd, jboolean
allowDurableConnections);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`allowDurableConnections` - TRUE if durable connections have to be enabled and FALSE otherwise

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_areDurableConnectionsAllowed

Description

Gives the status whether durable connection are allowed or not for this connection factory.

Function Signature

```
jint ucfmetadata_areDurableConnectionsAllowed(FHandle cfmd, jbool-
ean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if durable connections are allowed, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_disableCSPStoredMessageSend

Description

Disables the sending of any pending messages in the client side store of all durable connections created using this ConnectionFactory.

Function Signature

```
jint ucfmetadata_disableCSPStoredMessageSend(FHandle cfmd, const
char dontSend);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`dontSend` - True if send stored messages has to be disabled else false

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_disablePing

Description

Disables the ping functionality for this ConnectionFactory.

Function Signature

```
jint ucfmetadata_disablePing(FHandle cfmd, jboolean ping);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`ping` - TRUE will disabel pinging, FALSE will disable it.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getAdminConnectionReconnectInterval

Description

Returns the value for the reconnect interval between two reconnect attempts made all durable connections created using this Connection Factory.

Function Signature

```
jint ucfmetadata_getAdminConnectionReconnectInterval(FHandle cfmd,
jlong interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - Will contain the reconnect interval after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getAutoDispatch

Description

Returns the value of the auto dispatch boolean for this Connection Factory.

Function Signature

```
jint ucfmetadata_getAutoDispatch(FHandle cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the value of the auto dispatch boolean as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getBackupConnectURL

Description

Returns the back up connect URL at position num.

Function Signature

```
jint ucfmetadata_getBackupConnectURL(FHandle cfmd, jint num, const
char *url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the backup URL.

`url` - Will contain backup connection URL for this Connection Factory at position num.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getBackupInetAddress

Description

Gets the InetAddress at position num of the BackURL list.

Function Signature

```
jint ucfmetadata_getBackupInetAddress(FHandle cfmd, jint num, FHan-
dle inetAddress);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the InetAddress.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getBackupPort

Description

Gets the port for URL at position num of the BackURL list.

Function Signature

```
jint ucfmetadata_getBackupPort(FHandle cfmd, jint num, jint backup-
Port);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the URL.

`backupPort` - Will contain the backup port number after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getBackupURLStrings

Description

Gets the semicolon seperated string of Backup URLs.

Function Signature

```
jint ucfmetadata_getBackupURLStrings(FHandle cfmd, const char *url-
String);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`urlString` - Will contain the semicolon seperated list of backup URLs.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getBatchTimeoutInterval

Description

Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint ucfmetadata_getBatchTimeoutInterval(FHandle cfmd, const char
*interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - Will contain the CSP syncing delay.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getCBREnabled

Description

Returns true if Content Based Routing is enabled for this Connection Factory and false otherwise.

Function Signature

```
jint ucfmetadata_getCBREnabled(FHandle cfmd, const char *cbrEn-
abled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`cbrEnabled` - Will contain true if CBR is enabled else false.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getClientProxyURL

Description

Returns Client Proxy URL being used for TCP based connections created using this ConnectionFactory.

Function Signature

```
jint ucfmetadata_getClientProxyURL(FHandle cfmd, const char *url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`url` - Will contain the client proxy URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getCompressionManager

Description

Gets the CompressionManager class name to be used for MessageCompression purposes.

Function Signature

```
jint ucfmetadata_getCompressionManager(FHandle cfmd, const char
*compressionManager);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`compressionManager` - Will contain the name of the compression manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getConnectionClientID

Description

Returns the ClientID for this MetaData

Function Signature

```
jint ucfmetadata_getConnectionClientID(FHandle cfmd, const char
*clientID);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`clientID` - Will contain the client ID.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getConnectURL

Description

Returns the URL of the FMQ server to which all connections created using this ConnectionFactory will be made.

Function Signature

```
jint ucfmetadata_getConnectURL(FHandle cfmd, const char *url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`url` - Will contain the URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getCreateLocalSocket

Description

Gets the value of the local socket boolean.

Function Signature

```
jint ucfmetadata_getCreateLocalSocket(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain true or false as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getCSPUpdateFrequency

Description

Gets the UpdateFrequency set for this connection factory.

Function Signature

```
jint ucfmetadata_getCSPUpdateFrequency(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the update frequency as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getDurableConnectionBaseDir

Description

Gets the Durable Connection base directory for this ConnectionFactory as a string.

Function Signature

```
jint ucfmetadata_getDurableConnectionBaseDir(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Path of the base directory that is being used for client side storage of messages.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getDurableConnectionReconnectInterval

Description

Returns the value for the reconnect interval between two reconnect attempts made all durable connections created using this Connection Factory.

Function Signature

```
jint ucfmetadata_getDurableConnectionReconnectInterval(FHandle
cfmd, jlong interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - Will contain the reconnect interval.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getFactoryMetadataParams

Description

Returns a Hashtable of all the properties for this ConnectionFactory.

Function Signature

```
jint ucfmetadata_getFactoryMetadataParams(FHandle cfmd, FHandle
ht);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`ht` - Will contain a hashtable of all the properties for the connection factory.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getHTTPProxyURL

Description

Returns HTTP Proxy URL being used for HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint ucfmetadata_getHTTPProxyURL(FHandle cfmd, const char *url)
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`url` - Will contain the URL of the HTTP proxy as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getInetAddress

Description

Gets the Inet Address of the Server on which the ConnectionFactory resides in the Server.

Function Signature

```
jint ucfmetadata_getInetAddress(FHandle cfmd, FHandle inetAddress);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`inetAddress` - Will contain the Inet address of the server.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getLMSEnabled

Description

Returns whether support for large messages is enabled or not

Function Signature

```
jint ucfmetadata_getLMSEnabled(FHandle cfmd, jboolean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if large message support is enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getLookUpPreferredServer

Description

Gets the value for the lookup preferred server boolean.

Function Signature

```
jint ucfmetadata_getLookUpPreferredServer(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain true or false as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getMaxAdminConnectionReconnectAttempts

Description

Returns the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server) that are made by a admin connection created using this Connection Factory, as string.

Function Signature

```
jint ucfmetadata_getMaxAdminConnectionReconnectAttempts(FHandle
cfmd, jint result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the maximum number of reconnect attempts.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getMaxDurableConnectionReconnectAttempts

Description

Returns the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server) that are made by a durable connection created using this Connection Factory, as string.

Function Signature

```
jint ucfmetadata_getMaxDurableConnectionReconnectAttempts(FHandle
cfmd, jint result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the maximum number of reconnect attempts.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getMaxSocketCreationTries

Description

Returns the value for the Maximum socket creation tries for all connections] made using this ConnectionFactory.

Function Signature

```
jint ucfmetadata_getMaxSocketCreationTries(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the maximum number of socket creation tries as string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getPort

Description

Gets the port of the Server on which the ConnectionFactory resides in the Server.

Function Signature

```
jint ucfmetadata_getPort(FHandle cfmd, jint result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the port number.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getProxyAuthenticationRealm

Description

Returns the Proxy Authentication realm being used on the proxy for HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint ucfmetadata_getProxyAuthenticationRealm(FHandle cfmd, const
char *result)
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain name of the proxy authentication realm being used, as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getProxyCredentials

Description

Returns the password for the proxy through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint ucfmetadata_getProxyCredentials(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the password of the proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getProxyPrincipal

Description

Returns the username for the proxy through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint ucfmetadata_getProxyPrincipal(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the username of the proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getProxyType

Description

Returns the proxy name through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint ucfmetadata_getProxyType(FHandle cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the name of the proxy.

Returns

`MQ_SUCCESS` if successful, MQ_FAILURE if error.

ucfmetadata_getPublishBehaviourInAutoRevalidation

Description

Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint ucfmetadata_getPublishBehaviourInAutoRevalidation(FHandle
cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the CSP sync-up delay as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getPublishWaitDuringCSPSyncp

Description

Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint ucfmetadata_getPublishWaitDuringCSPSyncp(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the CSP sync-up delay.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getSecurityManager

Description

Returns the SecurityManager for this ConnectionFactory. Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint ucfmetadata_getSecurityManager(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the name of the security manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getSecurityProtocol

Description

Returns the security protocol for this ConnectionFactory.

Function Signature

```
jint ucfmetadata_getSecurityProtocol(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the security protocol.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getServerProxyURL

Description

Returns Server Proxy URL being used for TCP based connections created using this ConnectionFactory.

Function Signature

```
jint ucfmetadata_getServerProxyURL(FHandle cfmd, const
char*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the URL of the server proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getShutdownHookEnabled

Description

Returns whether shutdown hook is enabled or not.

Function Signature

```
jint ucfmetadata_getShutdownHookEnabled(FHandle cfmd, jboolean
result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if shutdown hook is enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getSleepSocketCreationTries

Description

Returns the value for the sleep time between 2 socket creation tries for all connections] made using this ConnectionFactory.

Function Signature

```
jint ucfmetadata_getSleepSocketCreationTries(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the sleep time bewteen socket creation tries as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getSocketTimeout

Description

Returns the timeout value for the sockets of all connections created using this ConnectionFactory.

Function Signature

```
jint ucfmetadata_getSocketTimeout(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the timeout value as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getSOCKSProxyURL

Description

Returns SOCKS Proxy URL being used for HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint ucfmetadata_getSOCKSProxyURL(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the URI of the SOCKS proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getTCPBatchSize

Description

Gets the message batch size for this ConnectionFactory.

Function Signature

```
jint ucfmetadata_getTCPBatchSize(FHandle cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the batch size as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getTransportProtocol

Description

Returns the Transport Protocol for this ConnectionFactory.

Function Signature

```
jint ucfmetadata_getTransportProtocol(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the transport protocol being used as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getXASocketTimeout

Description

Returns the timeout value for the socket used for the XAResource prepare and commit calls.

Function Signature

```
jint ucfmetadata_getXASocketTimeout(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the timeout value as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_isAutoRevalidationEnabled

Description

Returns whether auto-revalidation is enabled or not.

Function Signature

```
jint ucfmetadata_isAutoRevalidationEnabled(FHandle cfmd, const
char*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain true if auto-revalidation is enabled, false otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_isConnectURLUpdationAllowed

Description

Indicates whether Connect URL can be updated or not.

Function Signature

```
jint ucfmetadata_isConnectURLUpdationAllowed(FHandle cfmd, jboolean
result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain a boolean indicating whether connect URL can be updated or not.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_isCSPStoredMessageSendDisabled

Description

Checks if send of pending messages is disabled for this connection factory.

Function Signature

```
jint ucfmetadata_isCSPStoredMessageSendDisabled(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - True if send pending is disabled else false; null is returned if no value was set.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_isPingDisabled

Description

Checks if ping is enabled or disabled for this ConnectionFactory.

Function Signature

```
jint ucfmetadata_isPingDisabled(FHandle cfmd, jboolean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if ping was disabled while creating the connection factory and FALSE if otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_isThreadContextClassLoaderUsed

Description

Gives value of USE_THREAD_CONTEXT_CLASS_LOADER.

Function Signature

```
jint ucfmetadata_isThreadContextClassLoaderUsed(FHandle cfmd,
jboolean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain boolean specifying whether USE_THREAD_CONTEXT_CLASS_LOADER is true or false.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_loadEnvParams

Description

Loads the properties from the Env that are not already specified in the metaData.

Function Signature

```
jint ucfmetadata_loadEnvParams(FHandle cfmd, FHandle env);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`env` - The Env properties that need to be loaded into the metaData.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setAdminConnectionReconnectInterval

Description

Sets the value for the reconnect interval which is the interval between two successive reconnect attempts that would be made by a Durable connection created using this Connection Factory.

Function Signature

```
jint ucfmetadata_setAdminConnectionReconnectInterval(FHandle cfmd,
const char interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - The reconnect interval as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setAutoDispatch

Description

Sets the value for the auto dispatch boolean.

Function Signature

```
jint ucfmetadata_setAutoDispatch(FHandle cfmd, const char autoDis-
patch);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`autoDispatch` - True or false passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setBackupConnectURL

Description

Sets the back up connect url at position num for this Connection Factory.

Function Signature

```
jint ucfmetadata_setBackupConnectURL(FHandle cfmd, jint num, const
char strBackupConnectURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the backup URL (0 for first backup URL).

`strBackupURL` - The backup URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setBackupConnectURLs

Description

Sets the Backup URL's.

Function Signature

```
jint ucfmetadata_setBackupConnectURLs(FHandle cfmd, const char
backupUrls);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`backupUrls` - A semicolon seperated list of URLs.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setBatchTimeoutInterval

Description

Sets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint ucfmetadata_setBatchTimeoutInterval(FHandle cfmd, const char
batchTimeoutInterval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`batchTimeoutInterval` - The delay in publishing new messages when CSP is syncing up.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setCBREnabled

Description

Enable Content Based Routing for all connections that will be created using this ConnectionFactory.

Function Signature

```
jint ucfmetadata_setCBREnabled(FHandle cfmd, const char isCBREn-
abled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isCBREnabled` - True to enable Content Based Routing else false, passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setClientProxyURL

Description

Sets the URL for the Client side Proxy through which TCP based Connections created using this ConnectionFactory will be routed using HTTP Tunneling.

Function Signature

```
jint ucfmetadata_setClientProxyURL(FHandle cfmd, const char proxyURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The client proxy URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setCompressionManager

Description

Sets the CompressionManager class name to be used for MessageCompresson purposes.

Function Signature

```
jint ucfmetadata_setCompressionManager(FHandle cfmd, const char nmanager);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`manager` - Name of the compression manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setConnectionClientID

Description

Sets the ClientID for this MetaData.

Function Signature

```
jint ucfmetadata_setConnectionClientID(FHandle cfmd, const char
clientID);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`clientID` - The client ID.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setConnectURL

Description

Sets the primary Connect URL for this ConnectionFactory.

Function Signature

```
jint ucfmetadata_setConnectURL(FHandle cfmd, const char strConnec-
tURL);)
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setCreateLocalSocket

Description

Sets the value for the local socket boolean.

Function Signature

```
jint ucfmetadata_setCreateLocalSocket(FHandle cfmd, const char
localSocket);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`localSocket` - True of false as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setCSPUpdateFrequency

Description

Sets the UpdateFrequency to the value passed.

Function Signature

```
jint ucfmetadata_setCSPUpdateFrequency(FHandle cfmd, const char
frequency);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`frequency` - The CSP update frequency.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setDurableConnectionBaseDir

Description

Sets the Durable Connection Base directory for this ConnectionFactory as a string.

Function Signature

```
jint ucfmetadata_setDurableConnectionBaseDir(FHandle cfmd, const
char durBaseDir);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`durBaseDir` - Fully qualified path of the base directory for client side storage of messages

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setDurableConnectionReconnectInterval

Description

Sets the value for the reconnect interval which is the interval between two successive reconnect attempts that would be made by a Durable connection created using this Connection Factory.

Function Signature

```
jint ucfmetadata_setDurableConnectionReconnectInterval(FHandle
cfmd, const char interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - The reconnect interval passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setFactoryMetadataParams

Description

Sets the Hashtable containing all properties for this Connection Factory.

Function Signature

```
jint ucfmetadata_setFactoryMetadataParams(FHandle cfmd, FHandle
params);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`params` - Hashtable containing the name value pairs of all Connection Factory properties.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setHTTPProxyURL

Description

Sets the URL for the HTTP Proxy through which HTTP based Connections created using this ConnectionFactory will be routed.

Function Signature

```
jint ucfmetadata_setHTTPProxyURL(FHandle cfmd, const char prox-
yURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The URL of the HTTP proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setLMSEnabled

Description

Enables/disables large message support.

Function Signature

```
jint ucfmetadata_setLMSEnabled(FHandle cfmd, jboolean isLMSEn-
abled);
```

Parameters

`cfmd` the connection factory meta data object to operate on.

`isLMSEnabled` TRUE if Large support has to be enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setLookUpPreferredServer

Description

Sets the value for the lookup preferred server boolean.

Function Signature

```
jint ucfmetadata_setLookUpPreferredServer(FHandle cfmd, const char
lookupPreferred);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`lookupPreferred` - True or false passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setMaxAdminConnectionReconnectAttempts

Description

Sets the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server) that will be made by an admin connection created using this Connection Factory.

Function Signature

```
jint ucfmetadata_setMaxAdminConnectionReconnectAttempts(FHandle
cfmd,const char numAttempts);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`numAttempts` - Maximum number of reconnect attempts passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setMaxDurableConnectionReconnectAttempts

Description

Sets the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server) that will be made by a durable connection created using this Connection Factory.

Function Signature

```
jint ucfmetadata_setMaxDurableConnectionReconnectAttempts(FHandle
cfmd, const char numAttempts);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`numAttempts` - Maximum number of reconnect attempts passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setMaxSocketCreationTries

Description

Sets the value for the maximum socket creation tries that the FioranoMQ runtime will make for creating the socket for all connections made using this Connection-Factory.

Function Signature

```
jint ucfmetadata_setMaxSocketCreationTries(FHandle cfmd, const char socketTries);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`socketTries` - The number of socket creation tries passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setProxyAuthenticationRealm

Description

Sets the Authentication Realm being used on the Proxy through which HTTP based Connections created using this ConnectionFactory will be routed.

Function Signature

```
jint ucfmetadata_setProxyAuthenticationRealm(FHandle cfmd, const char realm);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`realm` - Name of the proxy authentication realm to be used.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setProxyCredentials

Description

Sets the password for getting validated at the proxy (if required) which is to be used for the HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint ucfmetadata_setProxyCredentials(FHandle cfmd, const char pass-
word);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`password` - The password.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setProxyPrincipal

Description

Sets the username for the proxy (if required) through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint ucfmetadata_setProxyPrincipal(FHandle cfmd, const char user-
name);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`username` - The username.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setProxyType

Description

Sets the type (name) of the proxy being used for the HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint ucfmetadata_setProxyType(FHandle cfmd, const char proxy);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxy` - The type of the proxy as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setPublishBehaviourInAutoRevalidation

Description

Sets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint ucfmetadata_setPublishBehaviourInAutoRevalidation(FHandle
cfmd, const char publishBehaviour);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`publishBehaviour` - Publisher behaviour on auto-revalidation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setPublishWaitDuringCSPSyncp

Description

Sets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint ucfmetadata_setPublishWaitDuringCSPSyncp(FHandle cfmd, const
char delay);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`delay` - The delay passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setSecurityManager

Description

Sets the security manager that will be used to make SSL connections using this ConnectionFactory.

Function Signature

```
jint ucfmetadata_setSecurityManager(FHandle cfmd, const char man-
ager);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`manager` - The name of the security manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setSecurityProtocol

Description

Sets the security protocol over which all connections created using this ConnectionFactory communicate with the FMQ server

Function Signature

```
jint ucfmetadata_setSecurityProtocol(FHandle cfmd, const char pro-
tocol);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`protocol` - Security protocol, possible values are PHAOS_SSL and JSSE_SSL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setServerProxyURL

Description

Sets the URL for the Server side Proxy through which TCP based Connections created using this ConnectionFactory will be routed using HTTP Tunneling.

Function Signature

```
jint ucfmetadata_setServerProxyURL(FHandle cfmd, const char proxyURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The URL of the server proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setShutdownHookEnabled

Description

Enables/disables shutdown hook.

Function Signature

```
jint ucfmetadata_setShutdownHookEnabled(FHandle cfmd, jboolean isShutdownHookEnabled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isShutdownHookEnabled` - TRUE if shutdown hook had to be enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setSleepSocketCreationTries

Description

Sets the value for the sleep time bewteen 2 creation tries that the FioranoMQ runtime will make for creating the socket for all connections made using this ConnectionFactory.

Function Signature

```
jint ucfmetadata_setSleepSocketCreationTries(FHandle cfmd, const char sleepTime);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setSocketTimeout

Description

Sets the timeout value for the sockets of all connections created using this ConnectionFactory.

Function Signature

```
jint ucfmetadata_setSocketTimeout(FHandle cfmd, const char timeout);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`timeout` - The timeout value passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setSOCKSProxyURL

Description

Sets the URL for the SOCKS Proxy through which HTTP based Connections created using this ConnectionFactory will be routed.

Function Signature

```
jint ucfmetadata_setSOCKSProxyURL(FHandle cfmd, const char proxyURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The URL of the SOCKS proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setTCPBatchSize

Description

Sets the value of the message batch size that will be used for NP messages being sent using all connections created using this Connection Factory.

Function Signature

```
jint ucfmetadata_setTCPBatchSize(FHandle cfmd, const char batch-
Size);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`batchSize` - The message batch size that will be used for NP messages.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setTransportProtocol

Description

Sets the transport protocol over which all connections created using this ConnectionFactory communicate with the FMQ server.

Function Signature

```
jint ucfmetadata_setTransportProtocol(FHandle cfmd, const char pro-
tocol);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`protocol` - The protocol. Possible values are TCP or HTTP.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setXASocketTimeout

Description

Sets the timeout value for the socket when the XAResource prepare or commit call is made from the runtime.

Function Signature

```
jint ucfmetadata_setXASocketTimeout(FHandle cfmd, const char time-
out);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`timeout` - The timeout value as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_updateConnectURL

Description

Sets a boolean in ConnectionFactory indicating whether Connect URL can be updated or not

Function Signature

```
jint ucfmetadata_updateConnectURL(FHandle cfmd, jboolean flag);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`flag` - Indicates whether connect URL can be updated or not.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_useThreadContextClassLoader

Description

Sets the value of USE_THREAD_CONTEXT_CLASS_LOADER

Function Signature

```
jint ucfmetadata_useThreadContextClassLoader(FHandle cfmd, jboolean
isThreadContextClassLoaderUsed);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isThreadContextClassLoaderUsed` - TRUE or FALSE.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_validateMetaData

Description

Validates if the URL and the name have been specified properly or not.

Function Signature

```
jint ucfmetadata_validateMetaData(FHandle cfmd);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_validateURL

Description

Validates the URL to see if the URL is a IP address/hostname It replaces "local-host" with complete IP address.

Function Signature

```
jint ucfmetadata_validateURL(FHandle cfmd);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getDescription

Description

Gets the description of the Administered Object The object implements the read-External method to restore its contents by calling the methods of DataInput for primitive types and readObject for objects, strings and arrays.

Function Signature

```
jint ucfmetadata_getDescription(FHandle jmd, const char *descrip-
tion);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`description` - A description of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_getName

Description

Gets the Name of the Administered Object.

Function Signature

```
jint ucfmetadata_getName(FHandle jmd, const char *name);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`name` - The name of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_hashCode

Description

Returns a hash code value for the object.

Function Signature

```
jint ucfmetadata_hashCode(FHandle jmd, jint hc);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`hc` - The has code value of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setDescription

Description

Sets the description of the MetaData Object

Function Signature

```
jint ucfmetadata_setDescriptionFHandle jmd, const char strDescrip-
tion);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`strDescription` - A description for this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ucfmetadata_setName

Description

Sets the Name of this MetaData Object

Function Signature

```
jint ucfmetadata_setName(FHandle jmd, const char strName);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`strName` - The name of this meta data object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

UnifiedXAConnectionFactoryMetaData

This class represents the MetaData information for a CommonConnectionFactory at it is stored in an LDAP directory. Objects of this class are used to create CommonConnectionFactory objects.

createUnifiedXAConnectionFactoryMetaData

Description

Creates a new UnifiedXAConnectionFactoryMetaData object.

Function Signature

```
jint createUnifiedXAConnectionFactoryMetaData(FHandle uxacfmd);
```

Parameters

`uxacfmd` - A newly created UnifiedXAConnectionFactoryMetaData object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_equals

Description

Compares two UnifiedXAConnectionFactoryMetaData objects for equality.

Function Signature

```
jint uxacfmetadata_equals(FHandle uxacfmd, FHandle obj, jboolean
result);
```

Parameters

`ucfmd` - The UnifiedXAConnectionFactoryMetaData object to operate on.

`result` - Will contain TRUE if the two objects are equal, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_allowAutoRevalidation

Description

Sets the value for auto-revalidation.

Function Signature

```
jint uxacfmetadata_allowAutoRevalidation(FHandle cfmd, const char
allowAutoRevalidation);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`allowAutoRevalidationtrue` - If auto-revalidation is required, false otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_allowDurableConnections

Description

Enables the durable connections for this ConnectionFactory.

Function Signature

```
jint uxacfmetadata_allowDurableConnections(FHandle cfmd, jboolean
allowDurableConnections);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`allowDurableConnections` - TRUE if durable connections have to be enabled and FALSE otherwise

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_areDurableConnectionsAllowed

Description

Gives the status whether durable connection are allowed or not for this connection factory.

Function Signature

```
jint uxacfmetadata_areDurableConnectionsAllowed(FHandle cfmd,
jboolean result)
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if durable connections are allowed, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_disableCSPStoredMessageSend

Description

Disables the sending of any pending messages in the client side store of all durable connections created using this ConnectionFactory.

Function Signature

```
jint uxacfmetadata_disableCSPStoredMessageSend(FHandle cfmd, const
char dontSend);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`dontSend` - True if send stored messages has to be disabled else false

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_disablePing

Description

Disables the ping functionality for this ConnectionFactory.

Function Signature

```
jint uxacfmetadata_disablePing(FHandle cfmd, jboolean ping);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`ping` - TRUE will disabel pinging, FALSE will disable it.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getAdminConnectionReconnectInterval

Description

Returns the value for the reconnect interval between two reconnect attempts made all durable connections created using this Connection Factory.

Function Signature

```
jint uxacfmetadata_getAdminConnectionReconnectInterval(FHandle
cfmd, jlong interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - Will contain the reconnect interval after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getAutoDispatch

Description

Returns the value of the auto dispatch boolean for this Connection Factory.

Function Signature

```
jint uxacfmetadata_getAutoDispatch(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the value of the auto dispatch boolean as a string.

Returns

`MQ_SUCCESS` if successful, MQ_FAILURE if error.

uxacfmetadata_getBackupConnectURL

Description

Returns the back up connect URL at position num.

Function Signature

```
jint uxacfmetadata_getBackupConnectURL(FHandle cfmd, jint num,
const char *url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the backup URL.

`Url` - Will contain backup connection URL for this Connection Factory at position num.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getBackupInetAddress

Description

Gets the InetAddress at position num of the BackURL list.

Function Signature

```
jint uxacfmetadata_getBackupInetAddress(FHandle cfmd, jint num,
FHandle inetAddress);
```

Parameters

`cfmd` the connection factory meta data object to operate on.

`num` the index of the InetAddress.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getBackupPort

Description

Gets the port for URL at position num of the BackURL list.

Function Signature

```
jint uxacfmetadata_getBackupPort(FHandle cfmd, jint num, jint back-
upPort);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the URL.

`backupPort` - Will contain the backup port number after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getBackupURLStrings

Description

Gets the semicolon seperated string of Backup URLs.

Function Signature

```
jint uxacfmetadata_getBackupURLStrings(FHandle cfmd, const char
*urlString);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`urlString` - Will contain the semicolon seperated list of backup URLs.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getBatchTimeoutInterval

Description

Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint uxacfmetadata_getBatchTimeoutInterval(FHandle cfmd, const char
*interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - Will contain the CSP syncing delay.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getCBREnabled

Description

Returns true if Content Based Routing is enabled for this Connection Factory and false otherwise.

Function Signature\

```
jint uxacfmetadata_getCBREnabled(FHandle cfmd, const char *cbrEn-
abled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`cbrEnabled` - Will contain true if CBR is enabled else false.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getClientProxyURL

Description

Returns Client Proxy URL being used for TCP based connections created using this ConnectionFactory.

Function Signature

```
jint uxacfmetadata_getClientProxyURL(FHandle cfmd, const char
*url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`url` - Will contain the client proxy URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getCompressionManager

Description

Gets the CompressionManager class name to be used for MessageCompression purposes.

Function Signature

```
jint uxacfmetadata_getCompressionManager(FHandle cfmd, const char
*compressionManager);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`compressionManager` - will contain the name of the compression manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getConnectionClientID

Description

Returns the ClientID for this MetaData

Function Signature

```
jint uxacfmetadata_getConnectionClientID(FHandle cfmd, const char
*clientID);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`@para clientID` - Will contain the client ID.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getConnectURL

Description

Returns the URL of the FMQ server to which all connections created using this ConnectionFactory will be made.

Function Signature

```
jint uxacfmetadata_getConnectURL(FHandle cfmd, const char *url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`Url` - Will contain the URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getCreateLocalSocket

Description

Gets the value of the local socket boolean.

Function Signature

```
jint uxacfmetadata_getCreateLocalSocket(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain true or false as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getCSPUpdateFrequency

Description

Gets the UpdateFrequency set for this connection factory.

Function Signature

```
jint uxacfmetadata_getCSPUpdateFrequency(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the update frequency as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getDurableConnectionBaseDir

Description

Gets the Durable Connection base directory for this ConnectionFactory as a string.

Function Signature

```
jint uxacfmetadata_getDurableConnectionBaseDir(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Path of the base directory that is being used for client side storage of messages.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getDurableConnectionReconnectInterval

Description

Returns the value for the reconnect interval between two reconnect attempts made all durable connections created using this Connection Factory.

Function Signature

```
jint uxacfmetadata_getDurableConnectionReconnectInterval(FHandle
cfmd, jlong interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - Will contain the reconnect interval.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getFactoryMetadataParams

Description

Returns a Hashtable of all the properties for this ConnectionFactory.

Function Signature

```
jint uxacfmetadata_getFactoryMetadataParams(FHandle cfmd, FHandle
ht);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`ht` - Will contain a hashtable of all the properties for the connection factory.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getHTTPProxyURL

Description

Returns HTTP Proxy URL being used for HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint uxacfmetadata_getHTTPProxyURL(FHandle cfmd, const char *url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`Url` - Will contain the URL of the HTTP proxy as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getInetAddress

Description

Gets the Inet Address of the Server on which the ConnectionFactory resides in the Server.

Function Signature

```
jint uxacfmetadata_getInetAddress(FHandle cfmd, FHandle inetAd-
dress);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`inetAddress` - Will contain the Inet address of the server.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getLMSEnabled

Description

Returns whether support for large messages is enabled or not

Function Signature

```
jint uxacfmetadata_getLMSEnabled(FHandle cfmd, jboolean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if large message support is enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getLookUpPreferredServer

Description

Gets the value for the lookup preferred server boolean.

Function Signature

```
jint uxacfmetadata_getLookUpPreferredServer(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain true or false as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getMaxAdminConnectionReconnectAttempts

Description

Returns the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server) that are made by a admin connection created using this Connection Factory, as string.

Function Signature

```
jint uxacfmetadata_getMaxAdminConnectionReconnectAttempts(FHandle
cfmd, jint result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the maximum number of reconnect attempts.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getMaxDurableConnectionReconnectAttempts

Description

Returns the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server) that are made by a durable connection created using this Connection Factory, as string.

Function Signature

```
jint uxacfmetadata_getMaxDurableConnectionReconnectAttempts(FHan-
dle cfmd, jint result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the maximum number of reconnect attempts.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getMaxSocketCreationTries

Description

Returns the value for the Maximum socket creation tries for all connections] made using this ConnectionFactory.

Function Signature

```
jint uxacfmetadata_getMaxSocketCreationTries(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the maximum number of socket creation tries as string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getPort

Description

Gets the port of the Server on which the ConnectionFactory resides in the Server.

Function Signature

```
jint uxacfmetadata_getPort(FHandle cfmd, jint result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the port number.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getProxyAuthenticationRealm

Description

Returns the Proxy Authentication realm being used on the proxy for HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint uxacfmetadata_getProxyAuthenticationRealm(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain name of the proxy authentication realm being used, as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getProxyCredentials

Description

Returns the password for the proxy through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint uxacfmetadata_getProxyCredentials(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the password of the proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getProxyPrincipal

Description

Returns the username for the proxy through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint uxacfmetadata_getProxyPrincipal(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the username of the proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getProxyType

Description

Returns the proxy name through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint uxacfmetadata_getProxyTypecfmetadata_getProxyType
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the name of the proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getPublishBehaviourInAutoRevalidation

Description

Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint uxacfmetadata_getPublishBehaviourInAutoRevalidation(FHandle
cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the CSP sync-up delay as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getPublishWaitDuringCSPSyncp

Description

Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint uxacfmetadata_getPublishWaitDuringCSPSyncp(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the CSP sync-up delay.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getSecurityManager

Description

Returns the SecurityManager for this ConnectionFactory. Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint uxacfmetadata_getSecurityManager(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the name of the security manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getSecurityProtocol

Description

Returns the security protocol for this ConnectionFactory.

Function Signature

```
jint uxacfmetadata_getSecurityProtocol(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the security protocol.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getServerProxyURL

Description

Returns Server Proxy URL being used for TCP based connections created using this ConnectionFactory.

Function Signature

```
jint uxacfmetadata_getServerProxyURL(FHandle cfmd, const
char*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the URL of the server proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getShutdownHookEnabled

Description

Returns whether shutdown hook is enabled or not.

Function Signature

```
jint uxacfmetadata_getShutdownHookEnabled(FHandle cfmd, jboolean
result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if shutdown hook is enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getSleepSocketCreationTries

Description

Returns the value for the sleep time between 2 socket creation tries for all connections] made using this ConnectionFactory.

Function Signature

```
jint uxacfmetadata_getSleepSocketCreationTries(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the sleep time bewteen socket creation tries as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getSocketTimeout

Description

Returns the timeout value for the sockets of all connections created using this ConnectionFactory.

Function Signature

```
jint uxacfmetadata_getSocketTimeout(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the timeout value as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getSOCKSProxyURL

Description

Returns SOCKS Proxy URL being used for HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint uxacfmetadata_getSOCKSProxyURL(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the URI of the SOCKS proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getTCPBatchSize

Description

Gets the message batch size for this ConnectionFactory.

Function Signature

```
jint uxacfmetadata_getTCPBatchSize(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the batch size as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getTransportProtocol

Description

Returns the Transport Protocol for this ConnectionFactory.

Function Signature

```
jint uxacfmetadata_getTransportProtocol(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the transport protocol being used as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getXASocketTimeout

Description

Returns the timeout value for the socket used for the XAResource prepare and commit calls.

Function Signature

```
jint uxacfmetadata_getXASocketTimeout(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the timeout value as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_isAutoRevalidationEnabled

Description

Returns whether auto-revalidation is enabled or not.

Function Signature

```
jint uxacfmetadata_isAutoRevalidationEnabled(FHandle cfmd, const
char*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain true if auto-revalidation is enabled, false otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_isConnectURLUpdationAllowed

Description

Indicates whether Connect URL can be updated or not.

Function Signature

```
jint uxacfmetadata_isConnectURLUpdationAllowed(FHandle cfmd, jbool-
ean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain a boolean indicating whether connect URL can be updated or not.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_isCSPStoredMessageSendDisabled

Description

Checks if send of pending messages is disabled for this connection factory.

Function Signature

```
jint uxacfmetadata_isCSPStoredMessageSendDisabled(FHandle cfmd,
const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - True if send pending is disabled else false; null is returned if no value was set.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_isPingDisabled

Description

Checks if ping is enabled or disabled for this ConnectionFactory.

Function Signature

```
jint uxacfmetadata_isPingDisabled(FHandle cfmd, jboolean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if ping was disabled while creating the connection factory and FALSE if otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_isThreadContextClassLoaderUsed

Description

Gives value of USE_THREAD_CONTEXT_CLASS_LOADER.

Function Signature

```
jint uxacfmetadata_isThreadContextClassLoaderUsed(FHandle cfmd,
jboolean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain boolean specifying whether USE_THREAD_CONTEXT_CLASS_LOADER is true or false.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_loadEnvParams

Description

Loads the properties from the Env that are not already specified in the metaData.

Function Signature

```
jint uxacfmetadata_loadEnvParams(FHandle cfmd, FHandle env);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`env` - The Env properties that need to be loaded into the metaData.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setAdminConnectionReconnectInterval

Description

Sets the value for the reconnect interval which is the interval between two successive reconnect attempts that would be made by a Durable connection created using this Connection Factory.

Function Signature

```
jint uxacfmetadata_setAdminConnectionReconnectInterval(FHandle
cfmd, const char interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - The reconnect interval as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setAutoDispatch

Description

Sets the value for the auto dispatch boolean.

Function Signature

```
jint uxacfmetadata_setAutoDispatch(FHandle cfmd, const char auto-
Dispatch);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`autoDispatch` - True or false passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setBackupConnectURL

Description

Sets the back up connect url at position num for this Connection Factory.

Function Signature

```
jint uxacfmetadata_setBackupConnectURL(FHandle cfmd, jint num,
const char strBackupConnectURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the backup URL (0 for first backup URL).

`strBackupURL` - The backup URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setBackupConnectURLs

Description

Sets the Backup URL's.

Function Signature

```
jint uxacfmetadata_setBackupConnectURLs(FHandle cfmd, const char
backupUrls);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`backupUrls` - A semicolon seperated list of URLs.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setBatchTimeoutInterval

Description

Sets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint uxacfmetadata_setBatchTimeoutInterval(FHandle cfmd, const char
batchTimeoutInterval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`batchTimeoutInterval` - The delay in publishing new messages when CSP is syncing up.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setCBREnabled

Description

Enable Content Based Routing for all connections that will be created using this ConnectionFactory.

Function Signature

```
jint uxacfmetadata_setCBREnabled(FHandle cfmd, const char isCBREnabled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isCBREnabled` - True to enable Content Based Routing else false, passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setClientProxyURL

Description

Sets the URL for the Client side Proxy through which TCP based Connections created using this ConnectionFactory will be routed using HTTP Tunneling.

Function Signature

```
jint uxacfmetadata_setClientProxyURL(FHandle cfmd, const char proxyURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The client proxy URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setCompressionManager

Description

Sets the CompressionManager class name to be used for MessageCompresson purposes.

Function Signature

```
jint uxacfmetadata_setCompressionManager(FHandle cfmd, const char
nmanager);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`manager` - Name of the compression manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setConnectionClientID

Description

Sets the ClientID for this MetaData.

Function Signature

```
jint uxacfmetadata_setConnectionClientID(FHandle cfmd, const char
clientID);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`clientID` - The client ID.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setConnectURL

Description

Sets the primary Connect URL for this ConnectionFactory.

Function Signature

```
jint uxacfmetadata_setConnectURL(FHandle cfmd, const char strCon-
nectURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setCreateLocalSocket

Description

Sets the value for the local socket boolean.

Function Signature

```
jint uxacfmetadata_setCreateLocalSocket(FHandle cfmd, const char
localSocket);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`localSocket` - True of false as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setCSPUpdateFrequency

Description

Sets the UpdateFrequency to the value passed.

Function Signature

```
jint uxacfmetadata_setCSPUpdateFrequency(FHandle cfmd, const char
frequency);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`frequency` - The CSP update frequency.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setDurableConnectionBaseDir

Description

Sets the Durable Connection Base directory for this ConnectionFactory as a string.

Function Signature

```
jint uxacfmetadata_setDurableConnectionBaseDir(FHandle cfmd, const
char durBaseDir);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`durBaseDir` - Fully qualified path of the base directory for client side storage of messages

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setDurableConnectionReconnectInterval

Description

Sets the value for the reconnect interval which is the interval between two successive reconnect attempts that would be made by a Durable connection created using this Connection Factory.

Function Signature

```
jint uxacfmetadata_setDurableConnectionReconnectInterval(FHandle
cfmd, const char interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - The reconnect interval passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setFactoryMetadataParams

Description

Sets the Hashtable containing all properties for this Connection Factory.

Function Signature

```
jint uxacfmetadata_setFactoryMetadataParams(FHandle cfmd, FHandle
params);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`params` - Hashtable containing the name value pairs of all Connection Factory properties.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setHTTPProxyURL

Description

Sets the URL for the HTTP Proxy through which HTTP based Connections created using this ConnectionFactory will be routed.

Function Signature

```
jint uxacfmetadata_setHTTPProxyURL(FHandle cfmd, const char prox-
yURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The URL of the HTTP proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setLMSEnabled

Description

Enables/disables large message support.

Function Signature

```
jint uxacfmetadata_setLMSEnabled(FHandle cfmd, jboolean isLMSEn-
abled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isLMSEnabled` - TRUE if Large support has to be enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setLookUpPreferredServer

Description

Sets the value for the lookup preferred server boolean.

Function Signature

```
jint uxacfmetadata_setLookUpPreferredServer(FHandle cfmd, const
char lookupPreferred);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`lookupPreferred` - True or false passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setMaxAdminConnectionReconnectAttempts

Description

Sets the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server) that will be made by an admin connection created using this Connection Factory.

Function Signature

```
jint uxacfmetadata_setMaxAdminConnectionReconnectAttempts(FHandle
cfmd,const char numAttempts);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`numAttempts` - Maximum number of reconnect attempts passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setMaxDurableConnectionReconnectAttempts

Description

Sets the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server) that will be made by a durable connection created using this Connection Factory.

Function Signature

```
jint uxacfmetadata_setMaxDurableConnectionReconnectAttempts(FHandle cfmd, const char numAttempts);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`numAttempts` - Maximum number of reconnect attempts passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setMaxSocketCreationTries

Description

Sets the value for the maximum socket creation tries that the FioranoMQ runtime will make for creating the socket for all connections made using this ConnectionFactory.

Function Signature

```
jint uxacfmetadata_setMaxSocketCreationTries(FHandle cfmd, const char socketTries);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`socketTries` - The number of socket creation tries passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setProxyAuthenticationRealm

Description

Sets the Authentication Realm being used on the Proxy through which HTTP based Connections created using this ConnectionFactory will be routed.

Function Signature

```
jint uxacfmetadata_setProxyAuthenticationRealm(FHandle cfmd, const
char realm);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`realm` - Name of the proxy authentication realm to be used.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setProxyCredentials

Description

Sets the password for getting validated at the proxy (if required) which is to be used for the HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint uxacfmetadata_setProxyCredentials(FHandle cfmd, const char
password);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`password` - The password.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setProxyPrincipal

Description

Sets the username for the proxy (if required) through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint uxacfmetadata_setProxyPrincipal(FHandle cfmd, const char username);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`username` - The username.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setProxyType

Description

Sets the type (name) of the proxy being used for the HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint uxacfmetadata_setProxyType(FHandle cfmd, const char proxy);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxy` - The type of the proxy as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setPublishBehaviourInAutoRevalidation

Description

Sets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint uxacfmetadata_setPublishBehaviourInAutoRevalidation(FHandle
cfmd, const char publishBehaviour);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`publishBehaviour` - Publisher behaviour on auto-revalidation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setPublishWaitDuringCSPSyncp

Description

Sets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint uxacfmetadata_setPublishWaitDuringCSPSyncp(FHandle cfmd, const
char delay);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`delay` - The delay passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setSecurityManager

Description

Sets the security manager that will be used to make SSL connections using this ConnectionFactory.

Function Signature

```
jint uxacfmetadata_setSecurityManager(FHandle cfmd, const char man-
ager);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`manager` - The name of the security manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setSecurityProtocol

Description

Sets the security protocol over which all connections created using this ConnectionFactory communicate with the FMQ server

Function Signature

```
jint uxacfmetadata_setSecurityProtocol(FHandle cfmd, const char protocol);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`protocol` - Security protocol, possible values are PHAOS_SSL and JSSE_SSL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setServerProxyURL

Description

Sets the URL for the Server side Proxy through which TCP based Connections created using this ConnectionFactory will be routed using HTTP Tunneling.

Function Signature

```
jint uxacfmetadata_setServerProxyURL(FHandle cfmd, const char proxyURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The URL of the server proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setShutdownHookEnabled

Description

Enables/disables shutdown hook.

Function Signature

```
jint uxacfmetadata_setShutdownHookEnabled(FHandle cfmd, jboolean
isShutdownHookEnabled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isShutdownHookEnabled` - TRUE if shutdown hook had to be enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setSleepSocketCreationTries

Description

Sets the value for the sleep time bewteen 2 creation tries that the FioranoMQ runtime will make for creating the socket for all connections made using this ConnectionFactory.

Function Signature

```
jint uxacfmetadata_setSleepSocketCreationTries(FHandle cfmd, const
char sleepTime);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setSocketTimeout

Description

Sets the timeout value for the sockets of all connections created using this ConnectionFactory.

Function Signature

```
jint uxacfmetadata_setSocketTimeout(FHandle cfmd, const char time-
out);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`timeout` - The timeout value passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setSOCKSProxyURL

Description

Sets the URL for the SOCKS Proxy through which HTTP based Connections created using this ConnectionFactory will be routed.

Function Signature

```
jint uxacfmetadata_setSOCKSProxyURL(FHandle cfmd, const char prox-
yURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The URL of the SOCKS proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setTCPBatchSize

Description

Sets the value of the message batch size that will be used for NP messages being sent using all connections created using this Connection Factory.

Function Signature

```
jint uxacfmetadata_setTCPBatchSize(FHandle cfmd, const char batch-
Size);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`batchSize` - The message batch size that will be used for NP messages.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setTransportProtocol

Description

Sets the transport protocol over which all connections created using this ConnectionFactory communicate with the FMQ server.

Function Signature

```
jint uxacfmetadata_setTransportProtocol(FHandle cfmd, const char protocol);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`protocol` - The protocol. Possible values are TCP or HTTP.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setXASocketTimeout

Description

Sets the timeout value for the socket when the XAResource prepare or commit call is made from the runtime.

Function Signature

```
jint uxacfmetadata_setXASocketTimeout(FHandle cfmd, const char timeout);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`timeout` - The timeout value as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_updateConnectURL

Description

Sets a boolean in ConnectionFactory indicating whether Connect URL can be updated or not

Function Signature

```
jint uxacfmetadata_updateConnectURL(FHandle cfmd, jboolean flag);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`flag` - Indicates whether connect URL can be updated or not.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_useThreadContextClassLoader

Description

Sets the value of USE_THREAD_CONTEXT_CLASS_LOADER

Function Signature

```
jint uxacfmetadata_useThreadContextClassLoader(FHandle cfmd, jbool-
ean isThreadContextClassLoaderUsed);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isThreadContextClassLoaderUsed` - TRUE or FALSE.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_validateMetaData

Description

Validates if the URL and the name have been specified properly or not.

Function Signature

```
jint uxacfmetadata_validateMetaData(FHandle cfmd);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_validateURL

Description

Validates the URL to see if the URL is a IP address/hostname It replaces "local-host" with complete IP address.

Function Signature

```
jint uxacfmetadata_validateURL(FHandle cfmd);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getDescription

Description

Gets the description of the Administered Object The object implements the read-External method to restore its contents by calling the methods of DataInput for primitive types and readObject for objects, strings and arrays.

Function Signature

```
jint uxacfmetadata_getDescription(FHandle jmd, const char *descrip-
tion);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`description` - A description of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_getName

Description

Gets the Name of the Administered Object.

Function Signature

```
jint uxacfmetadata_getName(FHandle jmd, const char *name);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`name` - The name of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_hashCode

Description

Returns a hash code value for the object.

Function Signature

```
jint uxacfmetadata_hashCode(FHandle jmd, jint hc);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`hc` - The has code value of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setDescription

Description

Sets the description of the MetaData Object

Function Signature

```
jint uxacfmetadata_setDescription(FHandle jmd, const char strDe-
scription);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`strDescription` - A description for this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

uxacfmetadata_setName

Description

Sets the Name of this MetaData Object

Function Signature

```
jint uxacfmetadata_setName(FHandle jmd, const char strName);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`strName` - The name of this meta data object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

XAQueueConnectionFactoryMetaData

This class represents the MetaData information for a XAQueueConnectionFactory at it is stored in an LDAP directory. Objects of this class are used to create XAQueueConnectionFactory objects. QueueConnectionFactoryMetaData serves as its base class and defines all the attributes of this object.

createXAQueueConnectionFactoryMetaData

Description

Creates a new XAQueueConnectionFactoryMetaData object.

Function Signature

```
jint createXAQueueConnectionFactoryMetaData(FHandle xaqcfmd);
```

Parameters

`xaqcfmd` - A newly created XAQueueConnectionFactoryMetaData object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_equals

Description

Compares two XAQueueConnectionFactoryMetaData objects for equality.

Function Signature

```
jint xaqcfmetadata_equals(FHandle xaqcfmd, FHandle obj, jboolean
result);
```

Parameters

`xaqcfmd` - The XAQueueConnectionFactoryMetaData object to operate on.

`result` - Will contain TRUE if the two objects are equal, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_allowAutoRevalidation

Description

Sets the value for auto-revalidation.

Function Signature

```
jint xaqcfmetadata_allowAutoRevalidation(FHandle cfmd, const char
allowAutoRevalidation);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`allowAutoRevalidationtrue` - If auto-revalidation is required, false otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_allowDurableConnections

Description

Enablesthe durable connections for this ConnectionFactory.

Function Signature

```
jint xaqcfmetadata_allowDurableConnections(FHandle cfmd, jboolean
allowDurableConnections);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`allowDurableConnections` - TRUE if durable connections have to be enabled and FALSE otherwise

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_areDurableConnectionsAllowed

Description

Gives the status whether durable connection are allowed or not for this connection factory.

Function Signature

```
jint xaqcfmetadata_areDurableConnectionsAllowedFHandle cfmd, jbool-
ean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if durable connections are allowed, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_disableCSPStoredMessageSend

Description

Disables the sending of any pending messages in the client side store of all durable connections created using this ConnectionFactory.

Function Signature

```
jint xaqcfmetadata_disableCSPStoredMessageSend(FHandle cfmd, const
char dontSend);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`dontSend` - True if send stored messages has to be disabled else false

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_disablePing

Description

Disables the ping functionality for this ConnectionFactory.

Function Signature

```
jint xaqcfmetadata_disablePing(FHandle cfmd, jboolean ping);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`ping` - TRUE will disabel pinging, FALSE will disable it.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getAdminConnectionReconnectInterval

Description

Returns the value for the reconnect interval between two reconnect attempts made all durable connections created using this Connection Factory.

Function Signature

```
jint xaqcfmetadata_getAdminConnectionReconnectInterval((FHandle
cfmd, jlong interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - Will contain the reconnect interval after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getAutoDispatch

Description

Returns the value of the auto dispatch boolean for this Connection Factory.

Function Signature

```
jint xaqcfmetadata_getAutoDispatch(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the value of the auto dispatch boolean as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getBackupConnectURL

Description

Returns the back up connect URL at position num.

Function Signature

```
jint xaqcfmetadata_getBackupConnectURL(FHandle cfmd, jint num,
const char *url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the backup URL.

`url` - Will contain backup connection URL for this Connection Factory at position num.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getBackupInetAddress

Description

Gets the InetAddress at position num of the BackURL list.

Function Signature

```
jint xaqcfmetadata_getBackupInetAddress(FHandle cfmd, jint num,
FHandle inetAddress);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the InetAddress.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getBackupPort

Description

Gets the port for URL at position num of the BackURL list.

Function Signature

```
jint xaqcfmetadata_getBackupPort(FHandle cfmd, jint num, jint back-
upPort);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the URL.

`backupPort` - Will contain the backup port number after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getBackupURLStrings

Description

Gets the semicolon seperated string of Backup URLs.

Function Signature

```
jint xaqcfmetadata_getBackupURLStrings(FHandle cfmd, const char
*urlString);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`urlString` - Will contain the semicolon seperated list of backup URLs.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getBatchTimeoutInterval

Description

Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint xaqcfmetadata_getBatchTimeoutInterval(FHandle cfmd, const char
*interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - Will contain the CSP syncing delay.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getCBREnabled

Description

Returns true if Content Based Routing is enabled for this Connection Factory and false otherwise.

Function Signature

```
jint xaqcfmetadata_getCBREnabled(FHandle cfmd, const char *cbrEn-
abled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`cbrEnabled` - Will contain true if CBR is enabled else false.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getClientProxyURL

Description

Returns Client Proxy URL being used for TCP based connections created using this ConnectionFactory.

Function Signature

```
jint xaqcfmetadata_getClientProxyURL(FHandle cfmd, const char
*url);
```

Parameters

`cfmd` - the connection factory meta data object to operate on.

`url` - Will contain the client proxy URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getCompressionManager

Description

Gets the CompressionManager class name to be used for MessageCompression purposes.

Function Signature

```
jint xaqcfmetadata_getCompressionManager(FHandle cfmd, const char
*compressionManager);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`compressionManager` - Will contain the name of the compression manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getConnectionClientID

Description

Returns the ClientID for this MetaData

Function Signature

```
jint xaqcfmetadata_getConnectionClientID(FHandle cfmd, const char
*clientID);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`@para clientID` - Will contain the client ID.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getConnectURL

Description

Returns the URL of the FMQ server to which all connections created using this ConnectionFactory will be made.

Function Signature

```
jint xaqcfmetadata_getConnectURL(FHandle cfmd, const char *url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`url` - Will contain the URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getCreateLocalSocket

Description

Gets the value of the local socket boolean.

Function Signature

```
jint xaqcfmetadata_getCreateLocalSocket(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain true or false as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getCSPUpdateFrequency

Description

Gets the UpdateFrequency set for this connection factory.

Function Signature

```
jint xaqcfmetadata_getCSPUpdateFrequency(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the update frequency as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getDurableConnectionBaseDir

Description

Gets the Durable Connection base directory for this ConnectionFactory as a string.

Function Signature

```
jint xaqcfmetadata_getDurableConnectionBaseDir(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Path of the base directory that is being used for client side storage of messages.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getDurableConnectionReconnectInterval

Description

Returns the value for the reconnect interval between two reconnect attempts made all durable connections created using this Connection Factory.

Function Signature

```
jint xaqcfmetadata_getDurableConnectionReconnectInterval(FHandle
cfmd, jlong interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - Will contain the reconnect interval.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getFactoryMetadataParams

Description

Returns a Hashtable of all the properties for this ConnectionFactory.

Function Signature

```
jint xaqcfmetadata_getFactoryMetadataParams(FHandle cfmd, FHandle
ht);
```

Parameters

`cfmd` -The connection factory meta data object to operate on.

`ht` - Will contain a hashtable of all the properties for the connection factory.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getHTTPProxyURL

Description

Returns HTTP Proxy URL being used for HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint xaqcfmetadata_getHTTPProxyURL(FHandle cfmd, const char *url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`url` - Will contain the URL of the HTTP proxy as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getInetAddress

Description

Gets the Inet Address of the Server on which the ConnectionFactory resides in the Server.

Function Signature

```
jint xaqcfmetadata_getInetAddress(FHandle cfmd, FHandle inetAd-
dress);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`inetAddress` - will contain the Inet address of the server.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getLMSEnabled

Description

Returns whether support for large messages is enabled or not

Function Signature

```
jint xaqcfmetadata_getLMSEnabled(FHandle cfmd, jboolean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if large message support is enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getLookUpPreferredServer

Description

Gets the value for the lookup preferred server boolean.

Function Signature

```
jint xaqcfmetadata_getLookUpPreferredServer(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain true or false as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getMaxAdminConnectionReconnectAttempts

Description

Returns the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server) that are made by a admin connection created using this Connection Factory, as string.

Function Signature

```
jint xaqcfmetadata_getMaxAdminConnectionReconnectAttempts(FHandle
cfmd, jint result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the maximum number of reconnect attempts.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getMaxDurableConnectionReconnectAttempts

Description

Returns the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server) that are made by a durable connection created using this Connection Factory, as string.

Function Signature

```
jint xaqcfmetadata_getMaxDurableConnectionReconnectAttempts(FHandle cfmd, jint result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the maximum number of reconnect attempts.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getMaxSocketCreationTries

Description

Returns the value for the Maximum socket creation tries for all connections] made using this ConnectionFactory.

Function Signature

```
jint xaqcfmetadata_getMaxSocketCreationTries(FHandle cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the maximum number of socket creation tries as string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cfmetadata_getPort

Description

Gets the port of the Server on which the ConnectionFactory resides in the Server.

Function Signature

```
jint xaqcfmetadata_getPort(FHandle cfmd, jint result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the port number.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getProxyAuthenticationRealm

Description

Returns the Proxy Authentication realm being used on the proxy for HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint xaqcfmetadata_getProxyAuthenticationRealm(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain name of the proxy authentication realm being used, as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getProxyCredentials

Description

Returns the password for the proxy through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint xaqcfmetadata_getProxyCredentials(FHandle cfmd, const char *re-
sult);
```

Parameters

`cfmd` - the connection factory meta data object to operate on.

`result` - Will contain the password of the proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getProxyPrincipal

Description

Returns the username for the proxy through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint xaqcfmetadata_getProxyPrincipal(FHandle cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the username of the proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getProxyType

Description

Returns the proxy name through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint xaqcfmetadata_getProxyType(FHandle cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the name of the proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getPublishBehaviourInAutoRevalidation

Description

Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint xaqcfmetadata_getPublishBehaviourInAutoRevalidation(FHandle
cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the CSP sync-up delay as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getPublishWaitDuringCSPSyncp

Description

Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint xaqcfmetadata_getPublishWaitDuringCSPSyncp(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the CSP sync-up delay.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getSecurityManager

Description

Returns the SecurityManager for this ConnectionFactory. Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint xaqcfmetadata_getSecurityManager(FHandle cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the name of the security manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getSecurityProtocol

Description

Returns the security protocol for this ConnectionFactory.

Function Signature

```
jint xaqcfmetadata_getSecurityProtocol(FHandle cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the security protocol.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getServerProxyURL

Description

Returns Server Proxy URL being used for TCP based connections created using this ConnectionFactory.

Function Signature

```
jint xaqcfmetadata_getServerProxyURL(FHandle cfmd, const char*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the URL of the server proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getShutdownHookEnabled

Description

Returns whether shutdown hook is enabled or not.

Function Signature

```
jint xaqcfmetadata_getShutdownHookEnabled(FHandle cfmd, jboolean
result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if shutdown hook is enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getSleepSocketCreationTries

Description

Returns the value for the sleep time between 2 socket creation tries for all connections made using this ConnectionFactory.

Function Signature

```
jint xaqcfmetadata_getSleepSocketCreationTries(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the sleep time bewteen socket creation tries as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getSocketTimeout

Description

Returns the timeout value for the sockets of all connections created using this ConnectionFactory.

Function Signature

```
jints xaqcfmetadata_getSocketTimeout(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the timeout value as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getSOCKSProxyURL

Description

Returns SOCKS Proxy URL being used for HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint xaqcfmetadata_getSOCKSProxyURL(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the URI of the SOCKS proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getTCPBatchSize

Description

Gets the message batch size for this ConnectionFactory.

Function Signature

```
jint xaqcfmetadata_getTCPBatchSize(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the batch size as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getTransportProtocol

Description

Returns the Transport Protocol for this ConnectionFactory.

Function Signature

```
jint xaqcfmetadata_getTransportProtocol(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the transport protocol being used as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getXASocketTimeout

Description

Returns the timeout value for the socket used for the XAResource prepare and commit calls.

Function Signature

```
jint xaqcfmetadata_getXASocketTimeout(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the timeout value as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_isAutoRevalidationEnabled

Description

Returns whether auto-revalidation is enabled or not.

Function Signature

```
jint xaqcfmetadata_isAutoRevalidationEnabled(FHandle cfmd, const
char*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain true if auto-revalidation is enabled, false otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_isConnectURLUpdationAllowed

Description

Indicates whether Connect URL can be updated or not.

Function Signature

```
jint xaqcfmetadata_isConnectURLUpdationAllowed(FHandle cfmd, jbool-
ean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain a boolean indicating whether connect URL can be updated or not.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_isCSPStoredMessageSendDisabled

Description

Checks if send of pending messages is disabled for this connection factory.

Function Signature

```
jint xaqcfmetadata_isCSPStoredMessageSendDisabled(FHandle cfmd,
const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Ttrue if send pending is disabled else false; null is returned if no value was set.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_isPingDisabled

Description

Checks if ping is enabled or disabled for this ConnectionFactory.

Function Signature

```
jint xaqcfmetadata_isPingDisabled(FHandle cfmd, jboolean result)
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if ping was disabled while creating the connection factory and FALSE if otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_isThreadContextClassLoaderUsed

Description

Gives value of USE_THREAD_CONTEXT_CLASS_LOADER.

Function Signature

```
jint xaqcfmetadata_isThreadContextClassLoaderUsed(FHandle cfmd,
jboolean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain boolean specifying whether USE_THREAD_CONTEXT_CLASS_LOADER is true or false.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_loadEnvParams

Description

Loads the properties from the Env that are not already specified in the metaData.

Function Signature

```
jint xaqcfmetadata_loadEnvParams(FHandle cfmd, FHandle env);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`env` - The Env properties that need to be loaded into the metaData.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setAdminConnectionReconnectInterval

Description

Sets the value for the reconnect interval which is the interval between two successive reconnect attempts that would be made by a Durable connection created using this Connection Factory.

Function Signature

```
jint xaqcfmetadata_setAdminConnectionReconnectInterval(FHandle
cfmd, const char interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - The reconnect interval as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setAutoDispatch

Description

Sets the value for the auto dispatch boolean.

Function Signature

```
jint xaqcfmetadata_setAutoDispatch(FHandle cfmd, const char auto-
Dispatch);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`autoDispatch` - True or false passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setBackupConnectURL

Description

Sets the back up connect url at position num for this Connection Factory.

Function Signature

```
jint xaqcfmetadata_setBackupConnectURL(FHandle cfmd, jint num,
const char strBackupConnectURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the backup URL (0 for first backup URL).

`strBackupURL` - The backup URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setBackupConnectURLs

Description

Sets the Backup URLs.

Function Signature

```
jint xaqcfmetadata_setBackupConnectURLs(FHandle cfmd, const char
backupUrls);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`backupUrls` - A semicolon seperated list of URLs.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setBatchTimeoutInterval

Description

Sets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint xaqcfmetadata_setBatchTimeoutInterval(FHandle cfmd, const char
batchTimeoutInterval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`batchTimeoutInterval` - The delay in publishing new messages when CSP is syncing up.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setCBREnabled

Description

Enable Content Based Routing for all connections that will be created using this ConnectionFactory.

Function Signature

```
jint xaqcfmetadata_setCBREnabled(FHandle cfmd, const char isCBREn-
abled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isCBREnabled` - True to enable Content Based Routing else false, passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setClientProxyURLh

Description

Sets the URL for the Client side Proxy through which TCP based Connections created using this ConnectionFactory will be routed using HTTP Tunneling.

Function Signature

```
jint xaqcfmetadata_setClientProxyURL(FHandle cfmd, const char prox-
yURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The client proxy URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setCompressionManager

Description

Sets the CompressionManager class name to be used for MessageCompresson purposes.

Function Signature

```
jint xaqcfmetadata_setCompressionManager(FHandle cfmd, const char
nmanager);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`manager` - Name of the compression manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setConnectionClientID

Description

Sets the ClientID for this MetaData.

Function Signature

```
jint xaqcfmetadata_setConnectionClientID(FHandle cfmd, const char
clientID);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`clientID` - The client ID.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setConnectURL\

Description

Sets the primary Connect URL for this ConnectionFactory.

Function Signature

```
jint xaqcfmetadata_setConnectURL(FHandle cfmd, const char strCon-
nectURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setCreateLocalSocket

Description

Sets the value for the local socket boolean.

Function Signature

```
jint xaqcfmetadata_setCreateLocalSocket(FHandle cfmd, const char
localSocket);)
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`localSocket` - True of false as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setCSPUpdateFrequency

Description

Sets the UpdateFrequency to the value passed.

Function Signature

```
jint xaqcfmetadata_setCSPUpdateFrequency(FHandle cfmd, const char
frequency);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`frequency` - The CSP update frequency.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setDurableConnectionBaseDir

Description

Sets the Durable Connection Base directory for this ConnectionFactory as a string.

Function Signature

```
jint xaqcfmetadata_setDurableConnectionBaseDir(FHandle cfmd, const
char durBaseDir);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`durBaseDir` - Fully qualified path of the base directory for client side storage of messages

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setDurableConnectionReconnectInterval

Description

Sets the value for the reconnect interval which is the interval between two successive reconnect attempts that would be made by a Durable connection created using this Connection Factory.

Function Signature

```
jint xaqcfmetadata_setDurableConnectionReconnectInterval(FHandle
cfmd, const char durBaseDir);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - The reconnect interval passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setFactoryMetadataParams

Description

Sets the Hashtable containing all properties for this Connection Factory.

Function Signature

```
jint xaqcfmetadata_setFactoryMetadataParams(FHandle cfmd, FHandle
params);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`params` - Hashtable containing the name value pairs of all Connection Factory properties.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setHTTPProxyURL

Description

Sets the URL for the HTTP Proxy through which HTTP based Connections created using this ConnectionFactory will be routed.

Function Signature

```
jint xaqcfmetadata_setHTTPProxyURL(FHandle cfmd, const char prox-
yURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The URL of the HTTP proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setLMSEnabled

Description

Enables/disables large message support.

Function Signature

```
jint xaqcfmetadata_setLMSEnabled(FHandle cfmd, jboolean isLMSEn-
abled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isLMSEnabled` - TRUE if Large support has to be enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setLookUpPreferredServer

Description

Sets the value for the lookup preferred server boolean.

Function Signature

```
jint xaqcfmetadata_setLookUpPreferredServer(FHandle cfmd, const
char lookupPreferred);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`lookupPreferred` - True or false passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setMaxAdminConnectionReconnectAttempts

Description

Sets the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server) that will be made by an admin connection created using this Connection Factory.

Function Signature

```
jint xaqcfmetadata_setMaxAdminConnectionReconnectAttempts(FHandle
cfmd,const char numAttempts);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`numAttempts` - Maximum number of reconnect attempts passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setMaxDurableConnectionReconnectAttempts

Description

Sets the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server) that will be made by a durable connection created using this Connection Factory.

Function Signature

```
jint xaqcfmetadata_setMaxDurableConnectionReconnectAttempts(FHandle cfmd, const char numAttempts);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`numAttempts` - Maximum number of reconnect attempts passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setMaxSocketCreationTries

Description

Sets the value for the maximum socket creation tries that the FioranoMQ runtime will make for creating the socket for all connections made using this ConnectionFactory.

Function Signature

```
jint xaqcfmetadata_setMaxSocketCreationTries(FHandle cfmd, const char socketTries);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`socketTries` - The number of socket creation tries passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setProxyAuthenticationRealm

Description

Sets the Authentication Realm being used on the Proxy through which HTTP based Connections created using this ConnectionFactory will be routed.

Function Signature

```
jint xaqcfmetadata_setProxyAuthenticationRealm(FHandle cfmd, const
char realm);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`realm` - Name of the proxy authentication realm to be used.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setProxyCredentials

Description

Sets the password for getting validated at the proxy (if required) which is to be used for the HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint xaqcfmetadata_setProxyCredentials(FHandle cfmd, const char
password);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`password` - The password.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setProxyPrincipal

Description

Sets the username for the proxy (if required) through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint xaqcfmetadata_setProxyPrincipal(FHandle cfmd, const char user-
name);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`username` - The username.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setProxyType

Description

Sets the type (name) of the proxy being used for the HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint xaqcfmetadata_setProxyType(FHandle cfmd, const char proxy);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxy` - The type of the proxy as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setPublishBehaviourInAutoRevalidation

Description

Sets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint xaqcfmetadata_setPublishBehaviourInAutoRevalidation(FHandle
cfmd, const char publishBehaviour);
```

Parameters

`cfmd` the connection factory meta data object to operate on.

`publishBehaviour` publisher behaviour on auto-revalidation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setPublishWaitDuringCSPSyncp

Description

Sets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint xaqcfmetadata_setPublishWaitDuringCSPSyncp(FHandle cfmd, const
char delay);
```

Parameters

`cfmd` the connection factory meta data object to operate on.

`delay` the delay passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setSecurityManager

Description

Sets the security manager that will be used to make SSL connections using this ConnectionFactory.

Function Signature

```
jint xaqcfmetadata_setSecurityManager(FHandle cfmd, const char man-
ager);
```

Parameters

`cfmd` the connection factory meta data object to operate on.

`manager` the name of the security manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setSecurityProtocol

Description

Sets the security protocol over which all connections created using this ConnectionFactory communicate with the FMQ server

Function Signature

```
jint xaqcfmetadata_setSecurityProtocol(FHandle cfmd, const char protocol);
```

Parameters

`cfmd` the connection factory meta data object to operate on.

`protocol` security protocol, possible values are PHAOS_SSL and JSSE_SSL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setServerProxyURL

Description

Sets the URL for the Server side Proxy through which TCP based Connections created using this ConnectionFactory will be routed using HTTP Tunneling.

Function Signature

```
jint xaqcfmetadata_setServerProxyURL(FHandle cfmd, const char proxyURL);
```

Parameters

`cfmd` the connection factory meta data object to operate on.

`proxyURL` the URL of the server proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setShutdownHookEnabled

Description

Enables/disables shutdown hook.

Function Signature

```
jint xaqcfmetadata_setShutdownHookEnabled(FHandle cfmd, jboolean
isShutdownHookEnabled);
```

Parameters

`cfmd` the connection factory meta data object to operate on.

`isShutdownHookEnabled` TRUE if shutdown hook had to be enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setSleepSocketCreationTries

Description

Sets the value for the sleep time bewteen 2 creation tries that the FioranoMQ runtime will make for creating the socket for all connections made using this ConnectionFactory.

Function Signature

```
jint xaqcfmetadata_setSleepSocketCreationTries(FHandle cfmd, const
char sleepTime);
```

Parameters

`cfmd` the connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setSocketTimeout

Description

Sets the timeout value for the sockets of all connections created using this ConnectionFactory.

Function Signature

```
jint xaqcfmetadata_setSocketTimeout(FHandle cfmd, const char time-
out);
```

Parameters

`cfmd` the connection factory meta data object to operate on.

`timeout` the timeout value passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setSOCKSProxyURL

Description

Sets the URL for the SOCKS Proxy through which HTTP based Connections created using this ConnectionFactory will be routed.

Function Signature

```
jint xaqcfmetadata_setSOCKSProxyURL(FHandle cfmd, const char proxyURL);
```

Parameters

`cfmd` the connection factory meta data object to operate on.

`proxyURL` the URL of the SOCKS proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setTCPBatchSize

Description

Sets the value of the message batch size that will be used for NP messages being sent using all connections created using this Connection Factory.

Function Signature

```
jint xaqcfmetadata_setTCPBatchSize(FHandle cfmd, const char batchSize);
```

Parameters

`cfmd` the connection factory meta data object to operate on.

`batchSize` the message batch size that will be used for NP messages.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setTransportProtocol

Description

Sets the transport protocol over which all connections created using this ConnectionFactory communicate with the FMQ server.

Function Signature

```
jint xaqcfmetadata_setTransportProtocol(FHandle cfmd, const char
protocol);
```

Parameters

`cfmd` the connection factory meta data object to operate on.

`protocol` the protocol. Possible values are TCP or HTTP.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setXASocketTimeout

Description

Sets the timeout value for the socket when the XAResource prepare or commit call is made from the runtime.

Function Signature

```
jint xaqc\fmetadata_setXASocketTimeout(FHandle cfmd, const char
timeout);
```

Parameters

`cfmd` the connection factory meta data object to operate on.

`timeout` the timeout value as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_updateConnectURL

Description

Sets a boolean in ConnectionFactory indicating whether Connect URL can be updated or not

Function Signature

```
jint xaqcfmetadata_updateConnectURL(FHandle cfmd, jboolean flag);
```

Parameters

`cfmd` the connection factory meta data object to operate on.

`flag` indicates whether connect URL can be updated or not.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_useThreadContextClassLoader

Description

Sets the value of USE_THREAD_CONTEXT_CLASS_LOADER

Function Signature

```
jint xaqcfmetadata_useThreadContextClassLoader(FHandle cfmd, jbool-
ean isThreadContextClassLoaderUsed);
```

Parameters

`cfmd` the connection factory meta data object to operate on.

`isThreadContextClassLoaderUsed` TRUE or FALSE.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_validateMetaData

Description

Validates if the URL and the name have been specified properly or not.

Function Signature

```
jint xaqcfmetadata_validateMetaData(cfmd) (FHandle cfmd);(cfmd)
```

Parameters

`cfmd` the connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_validateURL

Description

Validates the URL to see if the URL is a IP address/hostname It replaces "local-host" with complete IP address.

Function Signature

```
jint xaqcfmetadata_validateURL(FHandle cfmd);
```

Parameters

`cfmd` the connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getDescription

Description

Gets the description of the Administered Object The object implements the read-External method to restore its contents by calling the methods of DataInput for primitive types and readObject for objects, strings and arrays.

Function Signature

```
jint xaqcfmetadata_getDescription(FHandle jmd, const char *descrip-
tion);
```

Parameters

`jmd -` The JMS meta data object to operate on.

`description` - Description of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_getName

Description

Gets the Name of the Administered Object.

Function Signature

```
jint xaqcfmetadata_getName(FHandle jmd, const char *name);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`name` - The name of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_hashCode

Description

Returns a hash code value for the object.

Function Signature

```
jint xaqcfmetadata_hashCode(FHandle jmd, jint hc);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`hc` - The has code value of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setDescription

Description

Sets the description of the MetaData Object

Function Signature

```
jint xaqcfmetadata_setDescription(FHandle jmd, const char strDe-
scription);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`strDescription` - Description for this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcfmetadata_setName

Description

Sets the Name of this MetaData Object

Function Signature

```
jint xaqcfmetadata_setName(FHandle jmd, const char strName);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`strName` - The name of this meta data object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

XATopicConnectionFactoryMetaData

This class represents the MetaData information for a TopicConnectionFactory at it is stored in an LDAP directory. Objects of this class are used to create TopicConnectionFactory objects.

createXATopicConnectionFactoryMetaData

Description

Creates a new XATopicConnectionFactoryMetaData object.

Function Signature

```
jint createXATopicConnectionFactoryMetaData(FHandle xatcfmd);
```

Parameters

`xatcfmd` a newly created XATopicConnectionFactoryMetaData object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_equals

Description

Compares two XATopicConnectionFactoryMetaData objects for equality.

Function Signature

```
jint xatcfmetadata_equals(FHandle xatcfmd, FHandle obj, jboolean
result);
```

Parameters

`xatcfmd` - The XATopicConnectionFactoryMetaData object to operate on.

`result` - Will contain TRUE if the two objects are equal, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_allowAutoRevalidation

Description

Sets the value for auto-revalidation.

Function Signature

```
jint xatcfmetadata_allowAutoRevalidation(FHandle cfmd, const char
allowAutoRevalidation);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`allowAutoRevalidationtrue` - If auto-revalidation is required, false otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_allowDurableConnections

Description

Enablesthe durable connections for this ConnectionFactory.

Function Signature

```
jint xatcfmetadata_allowDurableConnections(FHandle cfmd, jboolean
allowDurableConnections);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`allowDurableConnections` - TRUE if durable connections have to be enabled and FALSE otherwise

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_areDurableConnectionsAllowed

Description

Gives the status whether durable connection are allowed or not for this connection factory.

Function Signature

```
jint xatcfmetadata_areDurableConnectionsAllowed(FHandle cfmd,
jboolean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if durable connections are allowed, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_disableCSPStoredMessageSend

Description

Disables the sending of any pending messages in the client side store of all durable connections created using this ConnectionFactory.

Function Signature

```
jint xatcfmetadata_disableCSPStoredMessageSend(FHandle cfmd, const
char dontSend);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`dontSend` - True if send stored messages has to be disabled else false

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_disablePing

Description

Disables the ping functionality for this ConnectionFactory.

Function Signature

```
jint xatcfmetadata_disablePing(FHandle cfmd, jboolean ping);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`ping` - TRUE will disabel pinging, FALSE will disable it.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getAdminConnectionReconnectInterval

Description

Returns the value for the reconnect interval between two reconnect attempts made all durable connections created using this Connection Factory.

Function Signature

```
jint xatcfmetadata_getAdminConnectionReconnectInterval(FHandle
cfmd, jlong interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - Will contain the reconnect interval after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getAutoDispatch

Description

Returns the value of the auto dispatch boolean for this Connection Factory.

Function Signature

```
jint xatcfmetadata_getAutoDispatch(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the value of the auto dispatch boolean as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getBackupConnectURL

Description

Returns the back up connect URL at position num.

Function Signature

```
jint xatcfmetadata_getBackupConnectURL(FHandle cfmd, jint num,
const char *url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the backup URL.

`url` - Will contain backup connection URL for this Connection Factory at position num.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getBackupInetAddress

Description

Gets the InetAddress at position num of the BackURL list.

Function Signature

```
jint xatcfmetadata_getBackupInetAddress(FHandle cfmd, jint num,
FHandle inetAddress);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the InetAddress.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getBackupPort

Description

Gets the port for URL at position num of the BackURL list.

Function Signature

```
jint xatcfmetadata_getBackupPort(FHandle cfmd, jint num, jint back-
upPort);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the URL.

`backupPort` - Will contain the backup port number after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getBackupURLStrings

Description

Gets the semicolon seperated string of Backup URLs.

Function Signature

```
jint xatcfmetadata_getBackupURLStrings(FHandle cfmd, const char
*urlString);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`urlString` - Will contain the semicolon seperated list of backup URLs.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getBatchTimeoutInterval

Description

Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint xatcfmetadata_getBatchTimeoutInterval(FHandle cfmd, const char
*interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - Will contain the CSP syncing delay.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getCBREnabled

Description

Returns true if Content Based Routing is enabled for this Connection Factory and false otherwise.

Function Signature

```
jint xatcfmetadata_getCBREnabled(FHandle cfmd, const char *cbrEn-
abled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`cbrEnabled` - Will contain true if CBR is enabled else false.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getClientProxyURL

Description

Returns Client Proxy URL being used for TCP based connections created using this ConnectionFactory.

Function Signature

```
jint xatcfmetadata_getClientProxyURL(FHandle cfmd, const char
*url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`Url` - Will contain the client proxy URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getCompressionManager

Description

Gets the CompressionManager class name to be used for MessageCompression purposes.

Function Signature

```
jint xatcfmetadata_getCompressionManager(FHandle cfmd, const char
*compressionManager);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`compressionManager` - Will contain the name of the compression manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getConnectionClientID

Description

Returns the ClientID for this MetaData

Function Signature

```
jint xatcfmetadata_getConnectionClientID(FHandle cfmd, const char
*clientID);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`@para clientID` - Will contain the client ID.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getConnectURL

Description

Returns the URL of the FMQ server to which all connections created using this ConnectionFactory will be made.

Function Signature

```
jint xatcfmetadata_getConnectURL(FHandle cfmd, const char *url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`Url` - Will contain the URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getCreateLocalSocket

Description

Gets the value of the local socket boolean.

Function Signature

```
jint xatcfmetadata_getCreateLocalSocket(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain true or false as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getCSPUpdateFrequency

Description

Gets the UpdateFrequency set for this connection factory.

Function Signature

```
jint xatcfmetadata_getCSPUpdateFrequency(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the update frequency as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getDurableConnectionBaseDir

Description

Gets the Durable Connection base directory for this ConnectionFactory as a string.

Function Signature

```
jint xatcfmetadata_getDurableConnectionBaseDir(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Path of the base directory that is being used for client side storage of messages.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getDurableConnectionReconnectInterval

Description

Returns the value for the reconnect interval between two reconnect attempts made all durable connections created using this Connection Factory.

Function Signature

```
jint xatcfmetadata_getDurableConnectionReconnectInterval(FHandle
cfmd, jlong interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - Will contain the reconnect interval.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getFactoryMetadataParams

Description

Returns a Hashtable of all the properties for this ConnectionFactory.

Function Signature

```
jint xatcfmetadata_getFactoryMetadataParams(FHandle cfmd, FHandle
ht);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`ht` - Will contain a hashtable of all the properties for the connection factory.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getHTTPProxyURL

Description

Returns HTTP Proxy URL being used for HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint xatcfmetadata_getHTTPProxyURL(FHandle cfmd, const char *url);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`Url` - Will contain the URL of the HTTP proxy as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getInetAddress

Description

Gets the Inet Address of the Server on which the ConnectionFactory resides in the Server.

Function Signature

```
jint xatcfmetadata_getInetAddress(FHandle cfmd, FHandle inetAd-
dress);
```

Parameters

cfmd - The connection factory meta data object to operate on.

inetAddress - Will contain the Inet address of the server.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getLMSEnabled

Description

Returns whether support for large messages is enabled or not

Function Signature

```
jint xatcfmetadata_getLMSEnabled(FHandle cfmd, jboolean result);
```

Parameters

cfmd - The connection factory meta data object to operate on.

result - Will contain TRUE if large message support is enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getLookUpPreferredServer

Description

Gets the value for the lookup preferred server boolean.

Function Signature

```
jint xatcfmetadata_getLookUpPreferredServer(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain true or false as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getMaxAdminConnectionReconnectAttempts

Description

Returns the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server)that are made by a admin connection created using this Connection Factory, as string.

Function Signature

```
jint xatcfmetadata_getMaxAdminConnectionReconnectAttempts(FHandle
cfmd, jint result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the maximum number of reconnect attempts.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getMaxDurableConnectionReconnectAttempts

Description

Returns the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server) that are made by a durable connection created using this Connection Factory, as string.

Function Signature

```
jint xatcfmetadata_getMaxDurableConnectionReconnectAttempts(FHandle cfmd, jint result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the maximum number of reconnect attempts.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getMaxSocketCreationTries

Description

Returns the value for the Maximum socket creation tries for all connections] made using this ConnectionFactory.

Function Signature

```
jint xatcfmetadata_getMaxSocketCreationTries(FHandle cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the maximum number of socket creation tries as string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getPort

Description

Gets the port of the Server on which the ConnectionFactory resides in the Server.

Function Signature

```
jint xatcfmetadata_getPort(FHandle cfmd, jint result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the port number.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getProxyAuthenticationRealm

Description

Returns the Proxy Authentication realm being used on the proxy for HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint xatcfmetadata_getProxyAuthenticationRealm(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain name of the proxy authentication realm being used, as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getProxyCredentials

Description

Returns the password for the proxy through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint xatcfmetadata_getProxyCredentials(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the password of the proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getProxyPrincipal

Description

Returns the username for the proxy through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint xatcfmetadata_getProxyPrincipal(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the username of the proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getProxyType

Description

Returns the proxy name through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint xatcfmetadata_getProxyType(FHandle cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the name of the proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getPublishBehaviourInAutoRevalidation

Description

Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint xatcfmetadata_getPublishBehaviourInAutoRevalidation(FHandle
cfmd, const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the CSP sync-up delay as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getPublishWaitDuringCSPSyncp

Description

Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint xatcfmetadata_getPublishWaitDuringCSPSyncp(FHandle cfmd, const
char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the CSP sync-up delay.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getSecurityManager

Description

Returns the SecurityManager for this ConnectionFactory. Gets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint xatcfmetadata_getSecurityManager(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the name of the security manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getSecurityProtocol

Description

Returns the security protocol for this ConnectionFactory.

Function Signature

```
jint xatcfmetadata_getSecurityProtocol(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the security protocol.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getServerProxyURL

Description

Returns Server Proxy URL being used for TCP based connections created using this ConnectionFactory.

Function Signature

```
jint xatcfmetadata_getServerProxyURL(FHandle cfmd, const
char*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the URL of the server proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getShutdownHookEnabled

Description

Returns whether shutdown hook is enabled or not.

Function Signature

```
jint xatcfmetadata_getShutdownHookEnabled(FHandle cfmd, jboolean
result);
```

Parameters

cfmd - The connection factory meta data object to operate on.

result - Will contain TRUE if shutdown hook is enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getSleepSocketCreationTries

Description

Returns the value for the sleep time between 2 socket creation tries for all connections] made using this ConnectionFactory.

Function Signature

```
jint xatcfmetadata_getSleepSocketCreationTries(FHandle cfmd, const
char *result);
```

Parameters

cfmd - The connection factory meta data object to operate on.

result - Will contain the sleep time bewteen socket creation tries as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getSocketTimeout

Description

Returns the timeout value for the sockets of all connections created using this ConnectionFactory.

Function Signature

```
jint xatcfmetadata_getSocketTimeout(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the timeout value as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getSOCKSProxyURL

Description

Returns SOCKS Proxy URL being used for HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint xatcfmetadata_getSOCKSProxyURL(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the URI of the SOCKS proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getTCPBatchSize

Description

Gets the message batch size for this ConnectionFactory.

Function Signature

```
jint xatcfmetadata_getTCPBatchSize(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the batch size as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getTransportProtocol

Description

Returns the Transport Protocol for this ConnectionFactory.

Function Signature

```
jint xatcfmetadata_getTransportProtocol(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the transport protocol being used as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getXASocketTimeout

Description

Returns the timeout value for the socket used for the XAResource prepare and commit calls.

Function Signature

```
jint xatcfmetadata_getXASocketTimeout(FHandle cfmd, const char
*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain the timeout value as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_isAutoRevalidationEnabled

Description

Returns whether auto-revalidation is enabled or not.

Function Signature

```
jint xatcfmetadata_isAutoRevalidationEnabled(FHandle cfmd, const
char*result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain true if auto-revalidation is enabled, false otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_isConnectURLUpdationAllowed

Description

Indicates whether Connect URL can be updated or not.

Function Signature

```
jint xatcfmetadata_isConnectURLUpdationAllowed(FHandle cfmd, jbool-
ean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain a boolean indicating whether connect URL can be updated or not.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_isCSPStoredMessageSendDisabled

Description

Checks if send of pending messages is disabled for this connection factory.

Function Signature

```
jint xatcfmetadata_isCSPStoredMessageSendDisabled(FHandle cfmd,
const char *result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - True if send pending is disabled else false; null is returned if no value was set.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_isPingDisabled

Description

Checks if ping is enabled or disabled for this ConnectionFactory.

Function Signature

```
jint xatcfmetadata_isPingDisabled(FHandle cfmd, jboolean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain TRUE if ping was disabled while creating the connection factory and FALSE if otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_isThreadContextClassLoaderUsed

Description

Gives value of USE_THREAD_CONTEXT_CLASS_LOADER.

Function Signature

```
jint xatcfmetadata_isThreadContextClassLoaderUsed(FHandle cfmd,
jboolean result);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`result` - Will contain boolean specifying whether USE_THREAD_CONTEXT_CLASS_LOADER is true or false.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_loadEnvParams

Description

Loads the properties from the Env that are not already specified in the metaData.

Function Signature

```
jint xatcfmetadata_loadEnvParams(FHandle cfmd, FHandle env);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`env` - The Env properties that need to be loaded into the metaData.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setAdminConnectionReconnectInterval

Description

Sets the value for the reconnect interval which is the interval between two successive reconnect attempts that would be made by a Durable connection created using this Connection Factory.

Function Signature

```
jint xatcfmetadata_setAdminConnectionReconnectInterval(FHandle
cfmd, const char interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - The reconnect interval as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setAutoDispatch

Description

Sets the value for the auto dispatch boolean.

Function Signature

```
jint xatcfmetadata_setAutoDispatch(FHandle cfmd, const char auto-
Dispatch);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`autoDispatch` - True or false passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setBackupConnectURL

Description

Sets the back up connect url at position num for this Connection Factory.

Function Signature

```
jint xatcfmetadata_setBackupConnectURL(FHandle cfmd, jint num,
const char strBackupConnectURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`num` - The index of the backup URL (0 for first backup URL).

`strBackupURL` - The backup URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setBackupConnectURLs

Description

Sets the Backup URL's.

Function Signature

```
jint xatcfmetadata_setBackupConnectURLs(FHandle cfmd, const char
backupUrls);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`backupUrls` - A semicolon seperated list of URLs.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setBatchTimeoutInterval

Description

Sets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint xatcfmetadata_setBatchTimeoutInterval(FHandle cfmd, const char
batchTimeoutInterval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`batchTimeoutInterval` - The delay in publishing new messages when CSP is syncing up.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setCBREnabled

Description

Enables Content Based Routing for all connections that will be created using this ConnectionFactory.

Function Signature

```
jint xatcfmetadata_setCBREnabled(FHandle cfmd, const char isCBREn-
abled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isCBREnabled` - True to enable Content Based Routing else false, passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setClientProxyURL

Description

Sets the URL for the Client side Proxy through which TCP based Connections created using this ConnectionFactory will be routed using HTTP Tunneling.

Function Signature

```
jint xatcfmetadata_setClientProxyURL(FHandle cfmd, const char prox-
yURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The client proxy URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setCompressionManager

Description

Sets the CompressionManager class name to be used for MessageCompresson purposes.

Function Signature

```
jint xatcfmetadata_setCompressionManager(FHandle cfmd, const char
nmanager);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`manager` - Name of the compression manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setConnectionClientID

Description

Sets the ClientID for this MetaData.

Function Signature

```
jint xatcfmetadata_setConnectionClientID(FHandle cfmd, const char
clientID);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`clientID` - The client ID.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setConnectURL

Description

Sets the primary Connect URL for this ConnectionFactory.

Function Signature

```
jint xatcfmetadata_setConnectURL(FHandle cfmd, const char strCon-
nectURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setCreateLocalSocket

Description

Sets the value for the local socket boolean.

Function Signature

```
jint xatcfmetadata_setCreateLocalSocket(FHandle cfmd, const char
localSocket);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`localSocket` - True of false as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setCSPUpdateFrequency

Description

Sets the UpdateFrequency to the value passed.

Function Signature

```
jint xatcfmetadata_setCSPUpdateFrequency(FHandle cfmd, const char
frequency);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`frequency` - The CSP update frequency.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setDurableConnectionBaseDir

Description

Sets the Durable Connection Base directory for this ConnectionFactory as a string.

Function Signature

```
jint xatcfmetadata_setDurableConnectionBaseDir(FHandle cfmd, const
char durBaseDir);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`durBaseDir` - Fully qualified path of the base directory for client side storage of messages

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setDurableConnectionReconnectInterval

Description

Sets the value for the reconnect interval which is the interval between two successive reconnect attempts that would be made by a Durable connection created using this Connection Factory.

Function Signature

```
jint xatcfmetadata_setDurableConnectionReconnectInterval(FHandle
cfmd, const char interval);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`interval` - The reconnect interval passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setFactoryMetadataParams

Description

Sets the Hashtable containing all properties for this Connection Factory.

Function Signature

```
jint xatcfmetadata_setFactoryMetadataParams(FHandle cfmd, FHandle
params);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`params` - Hashtable containing the name value pairs of all Connection Factory properties.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setHTTPProxyURL

Description

Sets the URL for the HTTP Proxy through which HTTP based Connections created using this ConnectionFactory will be routed.

Function Signature

```
jint xatcfmetadata_setHTTPProxyURL(FHandle cfmd, const char proxyURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The URL of the HTTP proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setLMSEnabled

Description

Enables/disables large message support.

Function Signature

```
jint xatcfmetadata_setLMSEnabled(FHandle cfmd, jboolean isLMSEnabled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isLMSEnabled` - TRUE if Large support has to be enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setLookUpPreferredServer

Description

Sets the value for the lookup preferred server boolean.

Function Signature

```
jint xatcfmetadata_setLookUpPreferredServer(FHandle cfmd, const
char lookupPreferred);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`lookupPreferred` - True or false passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setMaxAdminConnectionReconnectAttempts

Description

Sets the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server) that will be made by an admin connection created using this Connection Factory.

Function Signature

```
jint xatcfmetadata_setMaxAdminConnectionReconnectAttempts(FHandle
cfmd,const char numAttempts);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`numAttempts` - Maximum number of reconnect attempts passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setMaxDurableConnectionReconnectAttempts

Description

Sets the value for the Maximum number of reconnect attempts (for reconnecting to the FMQ server) that will be made by a durable connection created using this Connection Factory.

Function Signature

```
jint xatcfmetadata_setMaxDurableConnectionReconnectAttempts(FHandle cfmd, const char numAttempts);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`numAttempts` - Maximum number of reconnect attempts passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setMaxSocketCreationTries

Description

Sets the value for the maximum socket creation tries that the FioranoMQ runtime will make for creating the socket for all connections made using this ConnectionFactory.

Function Signature

```
jint xatcfmetadata_setMaxSocketCreationTries(FHandle cfmd, const char socketTries);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`socketTries` - The number of socket creation tries passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setProxyAuthenticationRealm

Description

Sets the Authentication Realm being used on the Proxy through which HTTP based Connections created using this ConnectionFactory will be routed.

Function Signature

```
jint xatcfmetadata_setProxyAuthenticationRealm(FHandle cfmd, const
char realm);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`realm` - Name of the proxy authentication realm to be used.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setProxyCredentials

Description

Sets the password for getting validated at the proxy (if required) which is to be used for the HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint xatcfmetadata_setProxyCredentials(FHandle cfmd, const char
password);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`password` - The password.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setProxyPrincipal

Description

Sets the username for the proxy (if required) through which HTTP based connections created using this ConnectionFactory are routed.

Function Signature

```
jint xatcfmetadata_setProxyPrincipal(FHandle cfmd, const char user-
name);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`username` - The username.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setProxyType

Description

Sets the type (name) of the proxy being used for the HTTP based connections created using this ConnectionFactory.

Function Signature

```
jint xatcfmetadata_setProxyType(FHandle cfmd, const char proxy);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxy` - The type of the proxy as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setPublishBehaviourInAutoRevalidation

Description

Sets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint xatcfmetadata_setPublishBehaviourInAutoRevalidation(FHandle
cfmd, const char publishBehaviour);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`publishBehaviour` - Publisher behaviour on auto-revalidation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setPublishWaitDuringCSPSyncp

Description

Sets the delay in publishing new messages when CSP is syncing up.

Function Signature

```
jint xatcfmetadata_setPublishWaitDuringCSPSyncp(FHandle cfmd, const
char delay);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`delay` - The delay passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setSecurityManager

Description

Sets the security manager that will be used to make SSL connections using this ConnectionFactory.

Function Signature

```
jint xatcfmetadata_setSecurityManagerFHandle cfmd, const char man-
ager);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`manager` - The name of the security manager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setSecurityProtocol

Description

Sets the security protocol over which all connections created using this ConnectionFactory communicate with the FMQ server

Function Signature

```
jint xatcfmetadata_setSecurityProtocol(FHandle cfmd, const char protocol);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`protocol` - Security protocol, possible values are PHAOS_SSL and JSSE_SSL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setServerProxyURL

Description

Sets the URL for the Server side Proxy through which TCP based Connections created using this ConnectionFactory will be routed using HTTP Tunneling.

Function Signature

```
jint xatcfmetadata_setServerProxyURL(FHandle cfmd, const char proxyURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The URL of the server proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setShutdownHookEnabled

Description

Enables/disables shutdown hook.

Function Signature

```
jint xatcfmetadata_setShutdownHookEnabled(FHandle cfmd, jboolean
isShutdownHookEnabled);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isShutdownHookEnabled` - TRUE if shutdown hook had to be enabled, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setSleepSocketCreationTries

Description

Sets the value for the sleep time bewteen 2 creation tries that the FioranoMQ runtime will make for creating the socket for all connections made using this ConnectionFactory.

Function Signature

```
jint xatcfmetadata_setSleepSocketCreationTries(FHandle cfmd, const
char sleepTime);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setSocketTimeout

Description

Sets the timeout value for the sockets of all connections created using this ConnectionFactory.

Function Signature

```
jint xatcfmetadata_setSocketTimeout(FHandle cfmd, const char time-
out);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`timeout` - The timeout value passed as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setSOCKSProxyURL

Description

Sets the URL for the SOCKS Proxy through which HTTP based Connections created using this ConnectionFactory will be routed.

Function Signature

```
jint xatcfmetadata_setSOCKSProxyURL(FHandle cfmd, const char proxyURL);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`proxyURL` - The URL of the SOCKS proxy.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setTCPBatchSize

Description

Sets the value of the message batch size that will be used for NP messages being sent using all connections created using this Connection Factory.

Function Signature

```
jint xatcfmetadata_setTCPBatchSize(FHandle cfmd, const char batchSize);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`batchSize` - The message batch size that will be used for NP messages.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setTransportProtocol

Description

Sets the transport protocol over which all connections created using this ConnectionFactory communicate with the FMQ server.

Function Signature

```
jint xatcfmetadata_setTransportProtocol(FHandle cfmd, const char protocol);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`protocol` - The protocol. Possible values are TCP or HTTP.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setXASocketTimeout

Description

Sets the timeout value for the socket when the XAResource prepare or commit call is made from the runtime.

Function Signature

```
jint xatcfmetadata_setXASocketTimeout(FHandle cfmd, const char timeout);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`timeout` - The timeout value as a string.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_updateConnectURL

Description

Sets a boolean in ConnectionFactory indicating whether Connect URL can be updated or not

Function Signature

```
jint xatcfmetadata_updateConnectURL(FHandle cfmd, jboolean flag);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`flag` - Indicates whether connect URL can be updated or not.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_useThreadContextClassLoader

Description

Sets the value of USE_THREAD_CONTEXT_CLASS_LOADER

Function Signature

```
jint xatcfmetadata_useThreadContextClassLoader(FHandle cfmd, jbool-
ean isThreadContextClassLoaderUsed);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

`isThreadContextClassLoaderUsed` - TRUE or FALSE.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_validateMetaData

Description

Validates if the URL and the name have been specified properly or not.

Function Signature

```
jint xatcfmetadata_validateMetaData(FHandle cfmd);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_validateURL

Description

Validates the URL to see if the URL is a IP address/hostname It replaces "local-host" with complete IP address.

Function Signature

```
jint xatcfmetadata_validateURL(FHandle cfmd);
```

Parameters

`cfmd` - The connection factory meta data object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getDescription

Description

Gets the description of the Administered Object The object implements the read-External method to restore its contents by calling the methods of DataInput for primitive types and readObject for objects, strings and arrays.

Function Signature

```
jint xatcfmetadata_getDescription(FHandle jmd, const char *descrip-
tion);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`description` - A description of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_getName

Description

Gets the Name of the Administered Object.

Function Signature

```
jint xatcfmetadata_getName(FHandle jmd, const char *name);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`name` - The name of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_hashCode

Description

Returns a hash code value for the object.

Function Signature

```
jint xatcfmetadata_hashCode(FHandle jmd, jint hc);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`hc` - The has code value of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setDescription

Description

Sets the description of the MetaData Object

Function Signature

```
jint xatcfmetadata_setDescription(FHandle jmd, const char strDe-
scription);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`strDescription` - A description for this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcfmetadata_setName

Description

Sets the Name of this MetaData Object

Function Signature

```
jint xatcfmetadata_setName(FHandle jmd, const char strName);
```

Parameters

`jmd` - The JMS meta data object to operate on.

`strName` - The name of this meta data object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ServerMetaData

This class represents the MetaData information for a User as it is stored in an LDAP directory

createServerMetaData

Description

Creates a new ServerMetaData object.

Function Signature

```
jint createServerMetaData(FHandle smd);
```

Parameters

`smd` - A newly created ServerMetaData object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

servermetadata_equals

Description

Compares the given object with this object.

Function Signature

```
jint servermetadata_equals(FHandle smd, FHandle obj, jboolean
result);
```

Parameters

`smd` - The ServerMetaData object to operate on.

`obj` - The object to compare against.

`result` - Will contain TRUE if the two objects are equal, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

servermetadata_getBackupURLs

Description

Gets the backup urls of the Server (representing the Least loaded Server)

Function Signature

```
jint servermetadata_getBackupURLs(FHandle smd, const char *result);
```

Parameters

`smd` - The ServerMetaData object to operate on.

`resultresult` - Will contain the back URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

servermetadata_getMaxNoOfClientConnections

Description

Gets Max No of Client Connections.

Function Signature

```
jint servermetadata_getMaxNoOfClientConnections(FHandle smd, jint
result);
```

Parameters

`smd` - The ServerMetaData object to operate on.

`result` - Will contain the maximum number of client connections.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

servermetadata_getSecurityManager

Description

Sets security manager.

Function Signature

```
jint servermetadata_getSecurityManager(FHandle smd, const char
*result);
```

Parameters

`smd` - The ServerMetaData object to operate on.

`result` - Will contain the name of the SecurityManager implementation.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

servermetadata_getSecurityProtocol

Description

Gets security protocol.

Function Signature

```
jint servermetadata_getSecurityProtocol(FHandle smd, const char
*result);
```

Parameters

`smd` - The ServerMetaData object to operate on.

`result` - Will contain the name of the security protocol being used.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

servermetadata_getServerAdminURL

Description

Gets the Admin URL of the Server

Function Signature

```
jint servermetadata_getServerAdminURL(FHandle smd, FHandle result);
```

Parameters

`smd` - The ServerMetaData object to operate on.

`result` - Will contain a URL object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

servermetadata_getServerURL

Description

Gets the URL of the Server (representing the Least loaded Server).

Function Signature

```
jint servermetadata_getServerURL(FHandle smd, FHandle result);
```

Parameters

`smd` - The ServerMetaData object to operate on.

`result` - Will contain a URL object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

servermetadata_getTransportProtocol

Description

Gets transport protocol.

Function Signature

```
jint servermetadata_getTransportProtocol(FHandle smd, const char
*result);
```

Parameters

`smd` - The ServerMetaData object to operate on.

`result` - Will contain the name of the transport protocol being used.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

servermetadata_setBackupURLs

Description

Sets the backup URLs of the Server (representing the Least loaded Server)

Function Signature

```
jint servermetadata_setBackupURLs(FHandle smd, const char url);
```

Parameters

`smd` - The ServerMetaData object to operate on.

`Url` - The URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

servermetadata_setMaxNoOfClientConnections

Description

Sets the Max No of Client Connections.

Function Signature

```
jint servermetadata_setMaxNoOfClientConnections(FHandle smd, jint
clientConnection);
```

Parameters

`smd` - The ServerMetaData object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

servermetadata_setSecurityManager

Description

Sets security manager.

Function Signature

```
jint servermetadata_setSecurityManager(FHandle smd, const char man-
ager);
```

Parameters

`smd` - The ServerMetaData object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

servermetadata_setSecurityProtocol

Description

Sets security protocol.

Function Signature

```
jint servermetadata_setSecurityProtocol(FHandle smd, const char
protocol);
```

Parameters

`smd` - The ServerMetaData object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

servermetadata_setServerAdminURL

Description

Sets the Admin URL of the Server.

Function Signature

```
jint servermetadata_setServerAdminURL(FHandle smd, FHandle url);
```

Parameters

`smd` - The ServerMetaData object to operate on.

`Url` - The URL object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

servermetadata_setServerURL

Description

Sets the URL of the Server (representing the Least loaded Server).

Function Signature

```
jint servermetadata_setServerURL(FHandle smd, FHandle url);
```

Parameters

`smd` - The ServerMetaData object to operate on.

`Url` - The URL object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

servermetadata_setTransportProtocol

Description

Sets transport protocol.

Function Signature

```
jint servermetadata_setTransportProtocol(FHandle smd, const char
protocol);
```

Parameters

`smd` - The ServerMetaData object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

3

JMS Interface APIs

This chapter describes the JMS interface APIs that are used to create the various JMS components.

This package provides classes to create JMS components such as connection Factories, Queues, Topics, Publishers, Subscribers, Sender and Receivers. Please provide more info if required. The various classes in this package include

- ConnectionFactory
- QueueConnectionFactory
- TopicConnectionFactory
- Connection
- QueueConnection
- TopicConnection
- Session
- QueueSession
- TopicSession
- Destination
- Queue
- Topic
- MessageProducer
- QueueSender
- TopicPublisher
- QueueRequestor
- MessageConsumer
- QueueReceiver

- TopicSubscriber
- TemporaryTopic
- Message
- TextMessage
- BytesMessage
- XAConnectionFactory
- XAQueueConnectionFactory
- XATopicConnectionFactory
- XAConnection
- XAQueueConnection
- XATopicConnection
- XASession
- XAQueueSession
- XATopicSession
- XAResource
- Xid
- FioranoXid

ConnectionFactory

A ConnectionFactory object encapsulates a set of connection configuration parameters that has been defined by an administrator. A client uses it to create a connection with a JMS provider. A ConnectionFactory object is a JMS administered object and supports concurrent use.

cf_createConnection

Description

Creates a connection with default user identity.

Function Signature

```
jint cf_createConnection(FHandle cf, FHandle *c);
```

Parameters

`cf -` The queue connection factory to operate on.

`c -` A newly created connection.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

cf_createConnection_1

Description

Creates a queue connection with specified user identity.

Function Signature

```
jint cf_createConnection_1(FHandle cf, FHandle *c, const char
*username, const char *passwd);
```

Parameters

`cf -` The queue connection factory to operate on.

`c -` A newly created connection.

`username -` The caller's username.

`passwd -` The caller's passwd.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

QueueConnectionFactory

A client uses a QueueConnectionFactory object to create QueueConnection objects with a point-to-point JMS provider. QueueConnectionFactory can be used to create a QueueConnection, from which specialized queue-related objects can be created.

qcf_createQueueConnection

Description

Creates a queue connection with default user identity.

Function Signature

```
jint qcf_createQueueConnection(FHandle qcf, FHandle *qc);
```

Parameters

`qcf -` The queue connection factory to operate on.

`qc -` A newly created queue connection.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcf_createQueueConnection_1

Description

Creates a queue connection with specified user identity.

Function Signature

```
jint qcf_createQueueConnection_1(FHandle qcf, FHandle *qc, const
char *username, const char *passwd);
```

Parameters

`qcf -` The queue connection factory to operate on.

`qc -` A newly created queue connection.

`username -` The caller's username.

`passwd -` The caller's passwd.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qcf_createConnection_1

Description

Creates a queue connection with specified user identity.

Function Signature

```
jint qcf_createConnection_1)FHandle cf, FHandle *c, const char
*username, const char *passwd);
```

Parameters

`cf -` The queue connection factory to operate on.

`c -` A newly created queue connection.

`username -` The caller's username.

`password -` The caller's password.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

TopicConnectionFactory

The TopicConnectionFactory object to create TopicConnection objects with a publish/subscribe JMS provider. A TopicConnectionFactory can be used to create a TopicConnection, from which specialized topic-related objects can be created.

tcf_createTopicConnection

Description

Creates a topic connection with default user identity

Function Signature

```
jint tcf_createTopicConnection(FHandle tcf, FHandle *tc);
```

Parameters

`tcf -` The topic connection factory to operate on.

`tc -` A newly created topic connection.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcf_createTopicConnection_1

Description

Create a topic connection with specified user identity.

Function Signature

```
jint tcf_createTopicConnection_1(FHandle tcf, FHandle *tc, const
char *username, const char *passwd);
```

Parameters

`tcf -` The topic connection factory to operate on.

`tc -` A newly created topic connection.

`username -` The caller's username.

`passwd -` The caller's passwd.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcf_createConnection

Description

Creates a connection with default user identity.

Function Signature

```
jint tcf_createConnection(FHandle cf, FHandle *c);
```

Paramaters

`cf -` The queue connection factory to operate on.

`c -` A newly created queue connection.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

tcf_createConnection_1

Description

Creates a queue connection with specified user identity.

Function Signature

```
jint tcf_createConnection_1(FHandle cf, FHandle *c, const char
*username, const char *passwd);
```

Parameters

`cf -` The queue connection factory to operate on.

`c -` A newly created queue connection.

`username -` The caller's username.

`password -` The caller's password.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

Connection

A Connection object is a client's active connection to its JMS provider. It typically allocates provider resources outside the Java Virtual Machine (JVM). It encapsulates an open connection with a JMS provider. It typically represents an open TCP/IP socket between a client and the service provider software.

connection_start

Description

Starts (or restarts) a connection's delivery of incoming messages.

Function Signature

```
jint connection_start(FHandle connection);
```

Parameters

`connection` - The connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

connection_stop

Description

Used to temporarily stop a connection's delivery of incoming messages.

Function Signature

```
jint connection_stop(FHandle connection);
```

Parameters

`connection` - The connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

connection_close

Description

Closes this connection.

Function Signature

```
jint connection_close(FHandle connection);
```

Parameters

`connection` - The connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

connection_getClientID

Description

Gets the client identifier for this connection.

Function Signature

```
jint connection_getClientID(FHandle connection, const char **re-
sult);
```

Parameters

`connection -` The connection to operate on.

`result -` Contains the client ID of this connection.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

connection_setClientID

Description

Sets the client identifier for this connection.

Function Signature

```
jint connection_setClientID(FHandle connection, const char *clien-
tID);
```

Parameters

`connection -` The connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

connection_setExceptionListener

Description

Sets an exception listener for this connection.

Function Signature

```
jint connection_setExceptionListener(FHandle connection, OnExcep-
tionMethod listener);
```

Parameters

`connection -` The connection to operate on.

`listener -` A pointer to the callback function to invoke when an exception occurs.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

connection_setExceptionListener_1

Description

Sets an exception listener for this connection.

Function Signature

```
jint connection_setExceptionListener_1(FHandle connection, OnExcep-
tionMethodWithParam listener, void *param);
```

Parameters

`connection -` The connection to operate on.

`listener -` A pointer to the callback function to invoke when an exception occurs.

`param -` Parameter to be used in the callback function.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

connection_createSession

Description

Creates a Session object.

Function Signature

```
jint connection_createSession(FHandle connection, jboolean trans-
acted, jint acknowledgeMode, FHandle *result);
```

Parameters

`connection -` The connection to operate on.

`transacted -` If TRUE the session is transacted.

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

`result -` Contains a newly created Session object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

QueueConnection

A QueueConnection object is an active connection to a point-to-point JMS provider. A client uses a QueueConnection object to create one or more QueueSession objects for producing and consuming messages. A QueueConnection can be used to create a QueueSession from which specialized queue-related objects can be created.

queueconnection_createQueueSession

Description

Creates a queue session.

Function Signature

```
jint queueconnection_createQueueSession(FHandle qc, FHandle *qses-
sion, jboolean transacted, jint ack_mode);
```

Parameters

`qc -` The queue connection to operate on.

`qsession -` A newly created queue session.

`transacted -` If TRUE the session is transacted.

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

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queueconnection_start

Description

Starts (or restarts) a queue connection's delivery of incoming messages.

Function Signature

```
jint queueconnection_start(FHandle connection);
```

Parameters

`connection` - The queue connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queueconnection_stop

Description

Used to temporarily stop a queue connection's delivery of incoming messages.

Function Signature

```
jint queueconnection_stop(FHandle connection);
```

Parameters

`connection` - The queue connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queueconnection_close

Description

Closes this queue connection.

Function Signature

```
jint queueconnection_close(FHandle connection);
```

Parameters

`connection` - The queue connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queueconnection_getClientID

Description

Gets the client identifier for this connection.

Function Signature

```
jint queueconnection_getClientID(FHandle connection, const char
**result);
```

Parameters

`connection -` The connection to operate on.

`result -` Contains the client ID of this connection.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queueconnection_setClientID

Description

Sets the client identifier for this connection.

Function Signature

```
jint queueconnection_setClientID(FHandle connection, const char
*clientID);
```

Parameters

`connection -` The connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queueconnection_setExceptionListener

Description

Sets an exception listener for this connection.

Function Signature

```
jint queueconnection_setExceptionListener(FHandle connection, OnExceptionMethod listener);
```

Parameters

`connection -` The connection to operate on.

`listener -` A pointer to the callback function to invoke when an exception occurs.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queueconnection_setExceptionListener_1

Description

Sets an exception listener for this connection.

Function Signature

```
jint queueconnection_setExceptionListener_1(FHandle connection, OnExceptionMethodWithParam listener, void *param);
```

Parameters

`connection -` The connection to operate on.

`listener -` A pointer to the callback function to invoke when an exception occurs.

`param -` Parameter to be used in the callback function.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queueconnection_createSession

Description

Creates a Session object.

Function Signature

```
jint queueconnection_createSession(FHandle connection, jboolean transacted, jint acknowledgeMode, FHandle *result);
```

Parameters

`connection -` The connection to operate on.

`transacted -` If TRUE the session is transacted.

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

`result -` Contains a newly created Session object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

TopicConnection

A TopicConnection object is an active connection to a publish/subscribe JMS provider. A client uses a TopicConnection object to create one or more TopicSession objects for producing and consuming messages. A TopicConnection can be used to create a TopicSession, from which specialized topic-related objects can be created.

topicconnection_createTopicSession

Description

Create a topic session.

Function Signature

```
jint topicconnection_createTopicSession(FHandle tc, FHandle *ts,
jboolean transacted, jint ack_mode);
```

Parameters

`tc -` The topic connection to operate on.

`ts -` A newly created topic session.

`transacted -` If TRUE the session is transacted.

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

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicconnection_start

Description

Start (or restart) a topic connection's delivery of incoming messages.

Function Signature

```
jint topicconnection_start(FHandle connection);
```

Parameters

`connection` - The topic connection to operate on.

Returns

MQ_SUCCESS If successful, MQ_FAILURE if error.

topicconnection_stop

Description

Used to temporarily stop a topic connection's delivery of incoming messages.

Function Signature

```
jint topicconnection_stop(FHandle connection);
```

Parameters

`connection` - The topic connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicconnection_close

Description

Closes this topic connection.

Declarartion

```
jint topicconnection_close(FHandle connection);
```

Parameters

`connection` - The topic connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicconnection_getClientID

Description

Gets the client identifier for this connection.

Function Signature

```
jint topicconnection_getClientID(FHandle connection, const char
**result);
```

Parameters

`connection -` The connection to operate on.

`result -` Contains the client ID of this connection.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicconnection_setClientID

Description

Sets the client identifier for this connection.

Function Signature

```
jint topicconnection_setClientID(FHandle connection, const char
*clientID);
```

Parameters

`connection -` The connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicconnection_setExceptionListener

Description

Sets an exception listener for this connection.

Function Signature

```
jint topicconnection_setExceptionListener(FHandle connection, OnEx-
ceptionMethod listener);
```

Parameters

`connection -` The connection to operate on.

`listener -` A pointer to the callback function to invoke when an exception occurs.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicconnection_setExceptionListener_1

Description

Sets an exception listener for this connection.

Function Signature

```
jint topicconnection_setExceptionListener_1(FHandle connection,
OnExceptionMethod listener);
```

Parameters

`connection -` The connection to operate on.

`listener -` A pointer to the callback function to invoke when an exception occurs.

`param -` Parameter to be used in the callback function.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicconnection_createSession

Description

Creates a Session object.

Function Signature

```
jint topicconnection_createSession(FHandle connection, jboolean
transacted, jint acknowledgeMode, FHandle *result);
```

Parameters

`connection -` The connection to operate on.

`transacted -` If TRUE the session is transacted.

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

`result -` Contains a newly created Session object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

Session

A Session object is a single-threaded context for producing and consuming messages. Although it may allocate provider resources outside the Java virtual machine (JVM), it is considered a lightweight JMS object.

It provides a way to create Queue or Topic objects for those clients that need to dynamically manipulate provider-specific destination names.

Following are the details of fields used in the Session object

- `#define AUTO_ACKNOWLEDGE 1` With this acknowledgement mode, the session automatically acknowledges a client's receipt of a message when it has either successfully returned from a call to receive or the message listener it has called to process the message successfully returns.
- `#define CLIENT_ACKNOWLEDGE 2` With this acknowledgement mode, the client acknowledges a message by calling a message's acknowledge method.
- `#define DUPS_OK_ACKNOWLEDGE 3` This acknowledgement mode instructs the session to lazily acknowledge the delivery of messages.

session_close

Description

Since a provider may allocate some resources on behalf of a Session outside the JVM, clients should close them when they are not needed.

Function Signature

```
jint session_close(FHandle session);
```

Parameters

`session` - The session to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

session_commit

Description

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

Function Signature

```
jint session_commit(FHandle session);
```

Parameters

`session -` The session to operate on.

Returns

MQ_SUCCESS If successful, MQ_FAILURE if error.

session_createBytesMessage

Description

Creates a bytes message.

Function Signature

```
jint session_createBytesMessage(FHandle session, FHandle *msg);
```

Parameters

`session -` The session to operate on.

`msg -` A newly created bytes message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

session_createTextMessage

Description

Create a text message to be used to send a message containing a string.

Function Signature

```
jint session_createTextMessage(FHandle session, FHandle *txtmsg);
```

Parameters

`session -` The session to operate on.

`txtmsg -` A newly created text message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

session_createTextMessage_1

Description

Create an initialized text message to be used to send a message containing a string.

Function Signature

```
jint session_createTextMessage_1(FHandle session, FHandle *txtmsg,
const char *text);
```

Parameters

`session -` The session to operate on.

`txtmsg -` A newly created text message.

`text -` The string used to initialize this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

session_getTransacted

Description

Is the session in transacted mode?

Function Signature

```
jint session_getTransacted(FHandle session, jboolean *transacted);
```

Parameters

`session -` The session to operate on.

`transacted -` TRUE, if in transacted mode.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

session_recover

Description

Stop message delivery in this session, and restart sending messages with the oldest unacknowledged message.

Function Signature

```
jint session_recover(FHandle session);
```

Parameters

`session -` The session to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

session_rollback

Description

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

Function Signature

```
jint session_rollback(FHandle session);
```

Parameters

`session -` The session to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

session_setMessageListener

Description

Set the session's distinguished message listener.

Function Signature

```
jint session_setMessageListener(FHandle session, OnMessageMethod
messageListener);
```

Parameters

`session -` The session to operate on.

`messageListener -` The callback function to invoke when a message arrives.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

session_setMessageListener_1

Description

Sets the session's distinguished message listener.

Function Signature

```
jint session_setMessageListener_1(FHandle session, OnMessageMethod-
WithParam messageListener, void *param);
```

Parameters

`session -` The session to operate on.

`messageListener -` The callback function to invoke when a message arrives.

`param -` A parameter to be used in the callback function.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

session_createBrowser

Description

Creates a QueueBrowser object to peek at the messages on the specified destination, which must be a queue.

Function Signature

```
jint session_createBrowser(FHandle session, FHandle destination,
FHandle *result);
```

Parameters

`session -` The Session to operate on.

`destination -` The Destination to access.

`result -` Contains the newly created QueueBrowser object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

session_createBrowser_1

Description

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

Function Signature

```
jint session_createBrowser_1(FHandle session, FHandle destination,
const char *messageSelector, FHandle *result);
```

Parameters

`session -` The Session to operate on.

`destination -` The Destination to access.

`messageSelector -` Only messages with properties matching the message selector expression are delivered.

`result -` Contains the newly created QueueBrowser object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

session_createConsumer

Description

Creates a MessageConsumer object to receive messages from the specified destination.

Function Signature

```
jint session_createConsumer(FHandle session, FHandle destination,
FHandle *result);
```

Parameters

`session -` The Session to operate on.

`destination -` The Destination to access.

`result -` Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

session_createConsumer_1

Description

Creates a message consumer to the specified destination, using a message selector.

Function Signature

```
jint session_createConsumer_1(FHandle session, FHandle destination,
const char *messageSelector, FHandle *result);
```

Parameters

`session -` The Session to operate on.

`destination -` The Destination to access.

`messageSelector -` Only messages with properties matching the message selector expression are delivered.

`result -` Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

session_createConsumer_2

Description

Creates a message consumer to the specified destination, using a message selector.

Function Signature

```
jint session_createConsumer_2(FHandle session, FHandle destination,
const char *messageSelector, jboolean NoLocal, FHandle *result);
```

Parameters

`session -` The Session to operate on.

`destination -` The Destination to access.

`messageSelector -` Only messages with properties matching the message selector expression are delivered.

`NoLocal -` If set, inhibits the delivery of messages published by its own connection.

`result -` Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

session_createConsumer_3

Description

Creates a durable message consumer of the specified destination The subscription name is used to support durable subscription to topics.

Function Signature

```
jint session_createConsumer_3(FHandle session, FHandle destination,
const char *messageSelector, jboolean NoLocal, const char *sub-
scriptionName, FHandle *result);
```

Parameters

`session -` The Session to operate on.

`destination -` The Destination to access.

`messageSelector -` Only messages with properties matching the message selector expression are delivered.

`NoLocal -` If set, inhibits the delivery of messages published by its own connection.

`subscriptionName -` The name of the subscription. If the Destination is a queue, the JMS Provider shall set this string to null.

`result -` Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

session_createDestinationIdentity

Description

Creates a destination object given a Destination name and destinationType (JMS_QUEUE or JMS_TOPIC).

Function Signature

```
jint session_createDestinationIdentity(FHandle session, const char
*destinationName, jint destinationType, FHandle *result);
```

Parameters

`session -` The Session to operate on.

`destinationName-` The name of this Destination.

`destinationType -` The type of this Destination.

`result -` Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

session_createProducer

Description

Creates a MessageProducer for the specified destination

Function Signature

```
jint session_createProducer(FHandle session, FHandle destination,
FHandle *result);
```

Parameters

`session -` The Session to operate on.

`destination -` The Destination to access.

`result -` Contains the newly created MessageProducer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

session_createTemporaryDestination

Description

Creates a TemporaryDestination object.

Function Signature

```
jint session_createTemporaryDestination(FHandle session, jint des-
tinationType, FHandle *result);
```

Parameters

`session`

The Session to operate on.

`destinationType`

The type of this TemporaryDestination.

`result`

Contains the newly created TemporaryDestination object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

session_getMessageConsumer

Description

Gets a MessageConsumer object based on a Destination and subscriptionName pair.

Function Signature

```
jint session_getMessageConsumer(FHandle session, FHandle destina-
tion, const char *subscriptionName, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destination -` The Destination to access.

`subscriptionName` - A string used to identify the subscription.

`result -` Contains a MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

QueueSession

A QueueSession object provides methods for creating QueueReceiver, QueueSender, QueueBrowser, and TemporaryQueue objects. A QueueSession is used for creating Point-to-Point specific objects.

qsession_createBrowser

Description

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

Function Signature

```
jint qsession_createBrowser(FHandle qs, FHandle queue, FHandle *result);
```

Parameters

`qs`

The queue session to operate on.

`queue`

The queue to access.

`result`

Contains the newly created QueueBrowser object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qsession_createBrowser_1

Description

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

Function Signature

```
jint qsession_createBrowser_1(FHandle qs, FHandle queue, const char *messageSelector, FHandle *result);
```

Parameters

`qs` - The queue session to operate on.

`queue` - The queue to access.

`messageSelector` - Only messages with properties matching the message selector expression are delivered.

`result` - Contains the newly created QueueBrowser object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error occur.

qsession_createSender

Description

Create a queue sender to send messages to the specified queue.

Function Signature

```
jint qsession_createSender(FHandle qs, FHandle *qsender, FHandle
q);
```

Parameters

`qs` - The queue session to operate on.

`qsender` - A newly created queue sender.

`q` - The queue to access.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qsession_createQueue

Description

Creates a queue identity given a queue name.

Function Signature

```
jint qsession_createQueue(FHandle qsession, FHandle *queue, const
char *queueName);
```

Parameters

`qs` - The queue session to operate on.

`queue` - A newly created queue with the given name.

`queueName` - The name of this queue.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qsession_createTemporaryQueue

Description

Creates a temporary queue.

Function Signature

```
jint qsession_createTemporaryQueue(FHandle qs, FHandle *tempQueue);
```

Parameters

`qs` - The queue session to operate on.

`tempQueue` - A newly created temporary queue identity.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qsession_createReceiver

Description

Creates a QueueReceiver to receive messages from the specified queue.

Function Signature

```
jint qsession_createReceiver(FHandle qs, FHandle *qreceiver, FHan-
dle queue);
```

Parameters

`qs` - The queue session to operate on.

`qreceiver` - A newly created queue receiver.

`queue` - The queue to access.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qsession_createReceiver_1

Description

Creates a QueueReceiver to receive messages from the specified queue.

Function Signature

```
jint qsession_createReceiver_1(FHandle qs, FHandle *qreceiver, FHan-
dle queue, const char *messageSelector);
```

Parameters

`qs` - The queue session to operate on.

`qreceiver` - A newly created queue receiver.

`queue` - The queue to access.

`messageSelector` - Only messages with properties matching the message selector expression are delivered.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qsession_close

Description

Since a provider may allocate some resources on behalf of a Session outside the JVM, clients should close them when they are not needed.

Function Signature

```
jint qsession_close(FHandle session);
```

Parameters

`session` - The session to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qsession_commit

Description

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

Function Signature

```
jint qsession_commit(FHandle session);
```

Parameters

`session -` The session to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qsession_createBytesMessage

Description

Creates a bytes message.

Function Signature

```
jint qsession_createBytesMessage(FHandle session, FHandle *msg);
```

Parameters

`session -` The session to operate on.

`param msg -` A newly created bytes message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qsession_createTextMessage

Description

Creates a text message to be used to send a message containing a string.

Function Signature

```
jint qsession_createTextMessage(FHandle session, FHandle *txtmsg);
```

Parameters

`session -` The queue session to operate on.

`txtmsg -` A newly created text message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qsession_createTextMessage_1

Description

Creates an initialized text message to be used to send a message containing a string.

Function Signature

```
jint qsession_createTextMessage_1(FHandle session, FHandle *txtmsg,
const char *text);
```

Parameters

`session -` The session to operate on.

`txtmsg -` A newly created text message.

`text -` The string used to initialize this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qsession_getTransacted

Description

Is the session in transacted mode?

Function Signature

```
jint qsession_getTransacted(FHandle session, jboolean *transacted);
```

Parameters

`session -` The session to operate on.

`transacted -` TRUE, if in transacted mode.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qsession_recover

Description

Stops message delivery in this session, and restart sending messages with the oldest unacknowledged message.

Function Signature

```
jint qsession_recover(FHandle session);
```

Parameters

`session -` The session to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qsession_rollback

Description

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

Function Signature

```
jint qsession_rollback(FHandle session);
```

Parameters

`session -` The session to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qsession_setMessageListener

Description

Sets the session's distinguished message listener.

Function Signature

```
jint qsession_setMessageListener(FHandle session, OnMessageMethod
messageListener);
```

Parameters

`session -` The session to operate on.

`messageListener -` The callback function to invoke when a message arrives.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qsession_setMessageListener_1

Description

Sets the session's distinguished message listener.

Function Signature

```
jint qsession_setMessageListener_1(FHandle session, OnMes-
sageMethodWithParam messageListener, void *param);
```

Parameters

`session -` The session to operate on.

`messageListener -` The callback function to invoke when a message arrives.

`param -` A parameter to be used in the callback function.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qsession_createConsumer

Description

Creates a MessageConsumer object to receive messages from the specified destination.

Function Signature

```
jint qsession_createConsumer(FHandle session, FHandle destination,
FHandle *result);
```

Parameters

`session -` The Session to operate on.

`destination -` The Destination to access.

`result -` Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qsession_createConsumer_1

Description

Creates a message consumer to the specified destination, using a message selector.

Function Signature

```
jint qsession_createConsumer_1(FHandle session, FHandle destination, const char *messageSelector, FHandle *result);
```

Parameters

`session -` The Session to operate on.

`destination -` The Destination to access.

`messageSelector -` Only messages with properties matching the message selector expression are delivered.

`result -` Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qsession_createConsumer_2

Description

Creates a message consumer to the specified destination, using a message selector.

Function Signature

```
jint qsession_createConsumer_2(FHandle session, FHandle destination, const char *messageSelector, jboolean NoLocal, FHandle *result);
```

Parameters

`session -` The Session to operate on.

`destination -` The Destination to access.

`messageSelector -` Only messages with properties matching the message selector expression are delivered.

`NoLocal -` If set, inhibits the delivery of messages published by its own connection.

`result -` Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qsession_createConsumer_3

Description

Creates a durable message consumer of the specified destination The subscription name is used to support durable subscription to topics.

Function Signature

```
jint qsession_createConsumer_3(FHandle session, FHandle destina-
tion, const char *messageSelector, jboolean NoLocal, const char
*subscriptionName, FHandle *result);
```

Parameters

`session -` The Session to operate on.

`destination -` The Destination to access.

`messageSelector -` Only messages with properties matching the message selector expression are delivered.

`NoLocal -` If set, inhibits the delivery of messages published by its own connection.

`subscriptionName -` The name of the subscription. If the Destination is a queue, the JMS Provider shall set this string to null.

`result -` Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qsession_createDestinationIdentity

Description

Creates a destination object given a Destination name and destinationType (JMS_QUEUE or JMS_TOPIC).

Function Signature

```
jint qsession_createDestinationIdentity(FHandle session, const char
*destinationName, jint destinationType, FHandle *result);
```

Parameters

`session -` The Session to operate on.

`destinationName` - The name of this Destination.

`destinationType` - The type of this Destination.

`result` - Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qsession_createProducer

Description

Creates a MessageProducer for the specified destination

Function Signature

```
jint qsession_createProducer(FHandle session, FHandle destination,
FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destination` - The Destination to access.

`result` - Contains the newly created MessageProducer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qsession_createTemporaryDestination

Description

Creates a TemporaryDestination object.

Function Signature

```
jint qsession_createTemporaryDestination(FHandle session, jint des-
tinationType, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destinationType` - The type of this TemporaryDestination.

`result` - Contains the newly created TemporaryDestination object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

qsession_getMessageConsumer

Description

Gets a MessageConsumer object based on a Destination and subscriptionName pair.

Function Signature

```
jint qsession_getMessageConsumer(FHandle session, FHandle destination, const char *subscriptionName, FHandle *result);
```

Parameters

`session -` The Session to operate on.

`destination -` The Destination to access.

`subscriptionName -` A string used to identify the subscription.

`result -` Contains a MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

TopicSession

A TopicSession object provides methods for creating TopicPublisher, TopicSubscriber, and TemporaryTopic objects. It also provides a method for deleting its client's durable subscribers. A TopicSession is used for creating Pub/Sub specific objects.

topicsession_createDurableSubscriber

Description

Create a durable Subscriber to the specified topic.

Function Signature

```
jint topicsession_createDurableSubscriber(FHandle ts, FHandle *topicSubscriber, FHandle topic, const char *name);
```

Parameters

`ts -` The topic session to operate on.

`topicSubscriber` - A newly created topic subscriber.

`topic` - The topic to subscribe to.

`name` - The name used to identify this subscription.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsession_createDurableSubscriber_1

Description

Create a durable Subscriber to the specified topic.

Function Signature

```
jint topicsession_createDurableSubscriber_1(FHandle ts, FHandle
*topicSubscriber, FHandle topic, const char *name, const char *mes-
sageSelector, jint noLocal);
```

Parameters

`ts` - The topic session to operate on.

`topicSubscriber` - A newly created topic subscriber.

`topic` - The topic to subscribe to.

`name` - 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 by its own connection.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsession_createPublisher

Description

Create a Publisher for the specified topic.

Function Signature

```
jint topicsession_createPublisher(FHandle ts, FHandle *tp, FHandle
topic);
```

Parameters

`ts` - The topic session to operate on.

`tp` - A newly created topic publisher.

`topic` - The topic to publish to.

Returns

MQ_SUCCESS If successful, MQ_FAILURE if error.

topicsession_createSubscriber

Description

Create a non-durable Subscriber to the specified topic.

Function Signature

```
jint topicsession_createSubscriber(FHandle ts, FHandle *subscriber,
FHandle topic);
```

Parameters

`ts` - The topic session to operate on.

`topicSubscriber` - A newly created topic subscriber.

`topic` - The topic to subscribe to.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsession_createSubscriber_1

Description

Create a non-durable Subscriber to the specified topic.

Function Signature

```
jint topicsession_createSubscriber_1(FHandle ts, FHandle *sub-
scriber, FHandle topic, const char *messageSelector, jboolean noLo-
cal);
```

Parameters

`ts` - The topic session to operate on.

`topicSubscriber` - A newly created topic subscriber.

`topic` - The topic to subscribe to.

`messageSelector -` Only messages with properties matching the message selector expression are delivered.

`noLocal if set -` Inhibits the delivery of messages published by its own connection.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

createTemporaryTopic

Description

Create a temporary topic.

Function Signature

```
jint topicsession_createTemporaryTopic(FHandle ts, FHandle *tempTopic);
```

Parameters

`ts -` The topic session to operate on.

`tempTopic -` A newly created temporary topic identity.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsession_createTopic

Description

Create a topic identity given a topic name.

Function Signature

```
jint topicsession_createTopic(FHandle ts, FHandle *topic, const char *name);
```

Parameters

`ts -` The topic session to operate on.

`topic -` A newly created topic with the given name.

`name -` The name of this topic.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsession_unsubscribe

Description

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

Function Signature

```
jint topicsession_unsubscribe(FHandle ts, const char *name);
```

Parameters

`ts` - The topic session to operate on.

`name` - The name used to identify this subscription.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsession_close

Description

Since a provider may allocate some resources on behalf of a Session outside the JVM, clients should close them when they are not needed.

Function Signature

```
jint topicsession_close(FHandle session);
```

Parameters

`session` - The session to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsession_commit

Description

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

Function Signature

```
jint topicsession_commit(FHandle session);
```

Parameters

`session` - The session to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsession_createBytesMessage

Description

Creates a bytes message.

Function Signature

```
jint topicsession_createBytesMessage(FHandle session, FHandle
*msg);
```

Parameters

`session -` The session to operate on.

`msg` - A newly created bytes message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsession_createTextMessage

Description

Create a text message to be used to send a message containing a string.

Function Signature

```
jint topicsession_createTextMessage(FHandle session, FHandle *txt-
msg);
```

Parameters

`ts -` The topic session to operate on.

`txtmsg -` A newly created text message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsession_createTextMessage_1

Description

Creates an initialized text message to be used to send a message containing a string.

Function Signature

```
jint topicsession_createTextMessage_1(FHandle session, FHandle
*txtmsg, const char *text);
```

Parameters

`session -` The session to operate on.

`txtmsg -` A newly created text message.

`text -` The string used to initialize this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsession_getTransacted

Description

Is the session in transacted mode?

Function Signature

```
jint topicsession_getTransacted(FHandle session, jboolean *trans-
acted);
```

Parameters

`session -` The session to operate on.

`transacted -` TRUE, if in transacted mode.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsession_recover

Description

Stops message delivery in this session, and restart sending messages with the oldest unacknowledged message.

Function Signature

```
jint topicsession_recover(FHandle session);
```

Parameters

`session -` The session to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsession_rollback

Description

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

Function Signature

```
jint topicsession_rollback(FHandle session);
```

Parameters

`session -` The session to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsession_setMessageListener

Description

Sets the session's distinguished message listener.

Function Signature

```
jint topicsession_setMessageListener(FHandle session, OnMes-
sageMethod messageListener);
```

Parameters

`session -` The session to operate on.

`messageListener -` The callback function to invoke when a message arrives.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsession_setMessageListener_1

Description

Sets the session's distinguished message listener.

Function Signature

```
jint topicsession_setMessageListener_1(FHandle session, OnMes-
sageMethodWithParam messageListener, void *param);
```

Parameters

`session -` The session to operate on.

`messageListener -` The callback function to invoke when a message arrives.

`param -` A parameter to be used in the callback function.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsession_createConsumer

Description

Createsa MessageConsumer object to receive messages from the specified destination.

Function Signature

```
jint topicsession_createConsumer(FHandle session, FHandle destina-
tion, FHandle *result);
```

Parameters

`session -` The Session to operate on.

`destination -` The Destination to access.

`result -` Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsession_createConsumer_1

Description

Createsa message consumer to the specified destination, using a message selector.

Function Signature

```
jint topicsession_createConsumer_1(FHandle session, FHandle destination, const char *messageSelector, FHandle *result);
```

Parameters

`session -` The Session to operate on.

`destination -` The Destination to access.

`messageSelector -` Only messages with properties matching the message selector expression are delivered.

`result -` Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsession_createConsumer_2

Description

Creates a message consumer to the specified destination, using a message selector.

Function Signature

```
jint topicsession_createConsumer_2(FHandle session, FHandle destination, const char *messageSelector, jboolean NoLocal, FHandle *result);
```

Parameters

`session -` The Session to operate on.

`destination -` The Destination to access.

`messageSelector -` Only messages with properties matching the message selector expression are delivered.

`NoLocal -` If set, inhibits the delivery of messages published by its own connection.

`result -` Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsession_createConsumer_3

Description

Creates a durable message consumer of the specified destination The subscription name is used to support durable subscription to topics.

Function Signature

```
jint topicsession_createConsumer_3(FHandle session, FHandle destination, const char *messageSelector, jboolean NoLocal, const char *subscriptionName, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destination` - The Destination to access.

`messageSelector` - Only messages with properties matching the message selector expression are delivered.

`NoLocal` - If set, inhibits the delivery of messages published by its own connection.

`subscriptionName` - The name of the subscription. If the Destination is a queue, the JMS Provider shall set this string to null.

`result` - Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsession_createDestinationIdentity

Description

Createsa destination object given a Destination name and destinationType (JMS_QUEUE or JMS_TOPIC).

Function Signature

```
jint topicsession_createDestinationIdentity(FHandle session, const char *destinationName, jint destinationType, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destinationName` - The name of this Destination.

`destinationType` - The type of this Destination.

`result` - Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsession_createProducer

Description

Creates a MessageProducer for the specified destination

Function Signature

```
jint topicsession_createProducer(FHandle session, FHandle destina-
tion, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destination` - The Destination to access.

`result` - Contains the newly created MessageProducer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsession_createTemporaryDestination

Description

Creates a TemporaryDestination object.

```
jint topicsession_createTemporaryDestination(FHandle session, jint
destinationType, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destinationType` - The type of this TemporaryDestination.

`result` - Contains the newly created TemporaryDestination object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsession_getMessageConsumer

Description

Gets a MessageConsumer object based on a Destination and subscriptionName pair.

Function Signature

```
jint topicsession_getMessageConsumer(FHandle session, FHandle des-
tination, const char *subscriptionName, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destination` - The Destination to access.

`subscriptionName -` A string used to identify the subscription.

`result -` Contains a MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

Destination

A Destination object encapsulates a provider-specific address.

destination_getDestinationName

Description

Gets the name of this destination.

Function Signature

```
jint destination_getDestinationName(FHandle destination, const char
**result);
```

Parameters

`destination` - The destination object to operate on.

`result` - Contains the name of this Destination.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

destination_isQueue

Description

Returns TRUE if the destination is a Queue.

Function Signature

```
jint destination_isQueue(FHandle destination, jboolean *result);
```

Parameters

`destination` - The destination object to operate on.

`result` - Contains TRUE if this Destination is a queue.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

destination_isTopic

Description

Returns TRUE if the destination is a Topic.

Function Signature

```
jint destination_isTopic(FHandle destination, jboolean *result);
```

Parameters

`destination -` The destination object to operate on.

`result` - Contains TRUE if this Destination is a topic

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

Queue

A Queue object encapsulates a provider-specific queue name. It is the way a client specifies the identity of a queue to JMS API methods

queue_getQueueName

Description

Get the name of a queue.

Function Signature

```
jint queue_getQueueName(FHandle queue, const char **name);
```

Parameters

`queue` - The queue to operate on.

`name` - Contain the queue name after the call completes.

Returns

MQ_SUCCESS If successful, MQ_FAILURE if error.

queue_getDestinationName

Description

Gets the name of this destination.

Function Signature

```
jint queue_getDestinationName(FHandle destination, const char **re-
sult);
```

Parameters

`destination` - The Destination object to operate on.

`result` - Contains the name of this Destination.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queue_isQueue

Description

Returns TRUE if the destination is a Queue.

Function Signature

```
jint queue_isQueue(FHandle destination, jboolean *result);
```

Parameters

`destination` - The destination object to operate on.

`result` - Contains TRUE if this Destination is a queue.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queue_isTopic

Description

Returns TRUE if the destination is a Topic.

Function Signature

```
jint queue_isTopic(FHandle destination, jboolean *result);
```

Parameters

`destination` - The destination object to operate on.

`result` - Contains TRUE if this Destination is a queue.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

Topic

A Topic object encapsulates a provider-specific topic name. It is the way a client specifies the identity of a topic to JMS API methods.

topic_getTopicName

Description

Gets the name of a queue.

Function Signature

```
jint topic_getTopicName(FHandle topic, const charFunction Signature
*name);
```

Parameters

`queue` - The queue to operate on.

`name` - Contains the queue name after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topic_getDestinationName

Description

Gets the name of this destination.

Function Signature

```
jint topic_getDestinationName(FHandle destination, const char **re-
sult);
```

Parameters

`destination` - The destination object to operate on.

`result` - Contains the name of this Destination.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topic_isQueue

Description

Returns TRUE if the destination is a Queue.

Function Signature

```
jint topic_isQueue(FHandle destination, jboolean *result);
```

Parameters

`destination` - The destination object to operate on.

`result` - Contains TRUE if this Destination is a queue.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topic_isTopic

Description

Returns TRUE if the destination is a Topic.

Function Signature

```
jint topic_isTopic(FHandle destination, jboolean *result);
```

Parameters

`destination` - The destination object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

DeliveryMode

The delivery modes supported by the JMS API are PERSISTENT and NON_PERSISTENT.Clients use delivery mode to tell a JMS provider how to balance message transport reliability with throughput.

Delivery mode covers only the transport of the message to its destination.

`DeliveryMode_NON_PERSISTENT 1`

This is the lowest-overhead delivery mode because it does not require that the message be logged to stable storage. The level of JMS provider failure that causes a NON_PERSISTENT message to be lost is not defined.

`DeliveryMode_PERSISTENT 2`

This delivery mode instructs the JMS provider to log the message to stable storage as part of the client's send operation. Only a hard media failure should cause a PERSISTENT message to be lost.

MessageProducer

A client uses a MessageProducer object to send messages to a destination. A MessageProducer object is created by passing a Destination object to a message-producer creation method supplied by a session. MessageProducer is the parent interface for all message producers.

messageproducer_close

Description

Since a provider may allocate some resources on behalf of a MessageProducer outside the JVM, clients should close them when they are not needed.

Function Signature

```
jint messageproducer_close(FHandle mp);
```

Parameters

`mp` - The MessageProducer to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

messageproducer_getDeliveryMode

Description

Gets the producer's default delivery mode.

Function Signature

```
jint messageproducer_getDeliveryMode(FHandle mp, jint result);
```

Parameters

`mp` - The MessageProducer to operate on.

`result` - Contains the producer's default delivery mode.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

messageproducer_getDisableMessageID

Description

Gets an indication of whether message IDs are disabled.

Function Signature

```
jint messageproducer_getDisableMessageID(FHandle mp, jboolean re-
sult);
```

Parameters

`mp` - The MessageProducer to operate on.

`result` - Contains an indication of whether message IDs are disabled.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

messageproducer_getDisableMessageTimestamp

Description

Gets an indication of whether message timestamps are disabled.

Function Signature

```
jint messageproducer_getDisableMessageTimestamp(FHandle mp, jbool-
ean result);
```

Parameters

`mp` - The MessageProducer to operate on.

`result` - Contains an indication of whether message timestamps are disabled.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

messageproducer_getPriority

Description

Gets the producer's default priority.

Function Signature

```
jint messageproducer_getPriority(FHandle mp, jint result);
```

Parameters

`mp` - The MessageProducer to operate on.

`result` - Contains the message priority for this MessageProducer.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

messageproducer_getTimeToLive

Description

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

Function Signature

```
jint messageproducer_getTimeToLive(FHandle mp, jlong result);
```

Parameters

`mp` - The MessageProducer to operate on.

`result` - Contains the message time to live in milliseconds; zero is unlimited.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

messageproducer_setDeliveryMode

Description

Sets the producer's default delivery mode.

Function Signature

```
jint messageproducer_setDeliveryMode(FHandle mp, jint delivery-
Mode);
```

Parameters

`mp` - The MessageProducer to operate on.

`deliveryMode` - The message delivery mode for this message producer. Legal values are DeliveryMode_NON_PERSISTENT or DeliveryMode_PERSISTENT.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

messageproducer_setDisableMessageID

Description

Sets whether message IDs are disabled.

Function Signature

```
jint messageproducer_setDisableMessageID(FHandle mp, jboolean val-
ue);
```

Parameters

`mp` - The MessageProducer to operate on.

`value` - Indicates if message IDs are to be disabled.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

messageproducer_setDisableMessageTimestamp

Description

Sets whether message timestamps are disabled.

Function Signature

```
jint messageproducer_setDisableMessageTimestamp(FHandle mp, jbool-
ean value);
```

Parameters

`mp` - The MessageProducer to operate on.

`value` - Indicates if message timestamps are to be disabled.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

messageproducer_setPriority

Description

Sets the producer's default priority.

Function Signature

```
jint messageproducer_setPriority(FHandle mp, jint defaultPriority);
```

Parameters

`mp` - The MessageProducer to operate on.

`defaultPriority` - The message priority for this message producer. Priority must be a value between 0 and 9.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

messageproducer_setTimeToLive

Description

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

Function Signature

```
jint messageproducer_setTimeToLive(FHandle mp, jlong timeToLive);
```

Parameters

`mp` - The MessageProducer to operate on.

`timeToLive` - The message time to live in milliseconds; zero is unlimited.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

messageproducer_sendMessage

Description

Sends a message to a Destination for an unidentified message producer.

Function Signature

```
jint messageproducer_sendMessage(FHandle mp, FHandle destination,
FHandle message);
```

Parameters

`mp` - The MessageProducer to operate on.

`destination` - The Destination to send the message to.

`message` - The message to be sent.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

messageproducer_sendMessage_1

Description

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

Function Signature

```
jint messageproducer_sendMessage_1(FHandle mp, FHandle destination,
FHandle message, jint deliveryMode, jint priority, jlong timeToL-
ive);
```

Parameters

`mp` - The MessageProducer to operate on.

`destination` - The Destination to send the message to.

`message` - The message to be sent.

`deliveryMode` - The delivery mode to use.

`priority` - The priority for this message.

`timeToLive` - The message's lifetime (in milliseconds).

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

messageproducer_sendMessage_2

Description

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

Function Signature

```
jint messageproducer_sendMessage_2(FHandle mp, FHandle message);
```

Parameters

`mp` - The MessageProducer to operate on.

`message` - The message to be sent.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

messageproducer_sendMessage_3

Description

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

Function Signature

```
jint messageproducer_sendMessage_3(FHandle mp, FHandle message, jint
deliveryMode, jint priority, jlong timeToLive);
```

Parameters

`mp` - The MessageProducer to operate on.

`message` - The message to be sent.

`deliveryMode` -The delivery mode to use.

`priority` - The priority for this message.

`timeToLive` - The message's lifetime (in milliseconds).

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

QueueSender

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

queuesender_getQueue

Description

Gets the queue associated with a queue sender.

Function Signature

```
jint queuesender_getQueue(FHandle qsender, FHandle queue);
```

Parameters

`qsender -` The queue sender to operate on.

`queue` - Contains the queue object after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuesender_send

Description

Sends a message to the queue. Use the queue sender's default delivery mode, timeToLive and priority.

Function Signature

```
jint queuesender_send(FHandle qsender, FHandle message);
```

Parameters

`qsender` - The queue sender to operate on.

`message` - The message to be sent.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuesender_send_1

Description

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

Function Signature

```
jint queuesender_send_1(FHandle qsender, FHandle message, jint de-
liveryMode, jint priority, jlong timeToLive);
```

Parameters

`qsender` - The queue sender to operate on.

`message` - The message to be sent.

`deliveryMode` - The delivery mode to use.

`priority` - The priority for this message.

`timeToLive` - The message's lifetime (in milliseconds).

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuesender_send_2

Description

Send a message to a queue for an unidentified message producer. Use the QueueSender's default delivery mode, timeToLive and priority

Function Signature

```
jint queuesender_send_2(FHandle qsender, FHandle queue, FHandle mes-
sage);
```

Parameters

`qsender` - The queue sender to operate on.

`queue` - The queue that this message should be sent to.

`message` - The message to be sent.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuesender_send_3

Description

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

Function Signature

```
jint queuesender_send_3(FHandle qsender, FHandle queue, FHandle message, jint deliveryMode, jint priority, jlong timeToLive);
```

Parameters

`qsender` - The queue sender to operate on.

`queue` - The queue that this message should be sent to.

`message` - The message to be sent.

`deliveryMode` - The delivery mode to use.

`priority` - The priority for this message.

`timeToLive` - The message's lifetime (in milliseconds).

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuesender_close

Description

Since a provider may allocate some resources on behalf of a MessageProducer outside the JVM, clients should close them when they are not needed.

Function Signature

```
jint queuesender_close(FHandle mp);
```

Parameters

`mp` - The MessageProducer to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuesender_getDeliveryMode

Description

Gets the producer's default delivery mode.

Function Signature

```
jint queuesender_getDeliveryMode(FHandle mp, jint result);
```

Parameters

`mp` - The MessageProducer to operate on.

`result` - Contains the producer's default delivery mode.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuesender_getDisableMessageID

Description

Gets an indication of whether message IDs are disabled.

Function Signature

```
jint queuesender_getDisableMessageID(FHandle mp, jboolean result);
```

Parameters

`mp` - The MessageProducer to operate on.

`result` - Contains an indication of whether message IDs are disabled.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuesender_getDisableMessageTimestamp

Description

Gets an indication of whether message timestamps are disabled.

Function Signature

```
jint queuesender_getDisableMessageTimestamp(FHandle mp, jboolean
result);
```

Parameters

`mp` - The MessageProducer to operate on.

`result` - Contains an indication of whether message timestamps are disabled.

Returns

`MQ_SUCCESS` if successful, MQ_FAILURE if error.

queuesender_getPriority

Description

Gets the producer's default priority.

Function Signature

```
jint queuesender_getPriority(FHandle mp, jint result);
```

Parameters

`mp` - The MessageProducer to operate on.

`result` - Contains the message priority for this MessageProducer.

Returns

`MQ_SUCCESS` if successful, MQ_FAILURE if error.

queuesender_getTimeToLive

Description

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

Function Signature

```
jint queuesender_getTimeToLive(FHandle mp, jlong result);
```

Parameters

`mp` - The MessageProducer to operate on.

`result` - Contains the message time to live in milliseconds zero is unlimited.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuesender_setDeliveryMode

Description

Sets the producer's default delivery mode.

Function Signature

```
jint queuesender_setDeliveryMode(FHandle mp, jint deliveryMode);
```

Parameters

`mp` - The MessageProducer to operate on.

`deliveryMode` - The message delivery mode for this message producer. Legal values are DeliveryMode_NON_PERSISTENT or DeliveryMode_PERSISTENT.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuesender_setDisableMessageID

Description

Sets whether message IDs are disabled.

Function Signature

```
jint queuesender_setDisableMessageID(FHandle mp, jboolean value);
```

Parameters

`mp` - The MessageProducer to operate on.

`value` - Indicates if message IDs are to be disabled.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuesender_setDisableMessageTimestamp

Description

Sets whether message timestamps are disabled.

Function Signature

```
jint queuesender_setDisableMessageTimestamp(FHandle mp, jboolean
value);
```

Parameters

`mp` - The MessageProducer to operate on.

`value` - Indicates if message timestamps are to be disabled.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuesender_setPriority

Description

Sets the producer's default priority.

Function Signature

```
jint queuesender_setPriority(FHandle mp, jint defaultPriority);
```

Parameters

`mp` - The MessageProducer to operate on.

`defaultPriority` - The message priority for this message producer. Priority must be a value between 0 and 9.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuesender_setTimeToLive

Description

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

Function Signature

```
jint queuesender_setTimeToLive(FHandle mp, jlong timeToLive);
```

Parameters

`mp` - The MessageProducer to operate on.

`timeToLive` - The message time to live in milliseconds zero is unlimited.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuesender_sendMessage

Description

Sends a message to a Destination for an unidentified message producer.

Function Signature

```
jint queuesender_sendMessage(FHandle mp, FHandle destination, FHan-
dle message);
```

Parameters

`mp` - The MessageProducer to operate on.

`destination` - The Destination to send the message to.

`message` - The message to be sent.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuesender_sendMessage_1

Description

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

Function Signature

```
jint queuesender_sendMessage_1(FHandle mp, FHandle destination,
FHandle message, jint deliveryMode, jint priority, jlong timeToL-
ive);
```

Parameters

`mp` - The MessageProducer to operate on.

`destination` - The Destination to send the message to.

`message` - The message to be sent.

`deliveryMode` - The delivery mode to use.

`priority` - The priority for this message.

`timeToLive` - The message's lifetime (in milliseconds).

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuesender_sendMessageMessage_2

Description

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

Function Signature

```
jint queuesender_sendMessageMessage_2(FHandle mp, FHandle message);
```

Parameters

`mp` - The MessageProducer to operate on.

`message` - The message to be sent.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuesender_sendMessage_3

Description

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

Function Signature

```
jint queuesender_sendMessage_3(FHandle mp, FHandle message, jint de-
liveryMode, jint priority, jlong timeToLive);
```

Parameters

`mp` - The MessageProducer to operate on.

`message` - The message to be sent.

`deliveryMode` - The delivery mode to use.

`priority` - The priority for this message.

`timeToLive` - The message's lifetime (in milliseconds).

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

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.

topicpublisher_publish

Description

Publishes a Message to the topic Use the topics default delivery mode, timeToLive and priority.

Function Signature

```
jint topicpublisher_publish(FHandle topicpublisher, FHandle msg);
```

Parameters

`topicpublisher` -The topic publisher to operate on.

`msg` - The message to be sent.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicpublisher_publish_1

Description

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

Function Signature

```
jint topicpublisher_publish_1(FHandle topicpublisher, FHandle msg,
jint deliveryMode, jint priority, jlong timeToLive);
```

Parameters

`topicpublisher` - The topic publisher to operate on.

`msg` - The message to be sent.

`deliveryMode` - The delivery mode to use.

`priority` - The priority for this message.

`timeToLive` - The message's lifetime (in milliseconds).

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicpublisher_publish_2

Description

Publishes a Message to the topic Use the topics default delivery mode, timeToLive and priority.

Function Signature

```
jint topicpublisher_publish_2(FHandle topicpublisher, FHandle top-
ic, FHandle msg);
```

Parameters

`topicpublisher` - The topic publisher to operate on.

`topic` - The topic to publish this message to.

`msg` - The message to be sent.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicpublisher_publish_3

Description

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

Function Signature

```
jint topicpublisher_publish_3(FHandle topicpublisher, FHandle top-
ic, FHandle msg, jint deliveryMode, jint priority, jlong timeToL-
ive);
```

Parameters

`topicpublisher` - The topic publisher to operate on.

`topic` - The topic to publish this message to.

`msg` - The message to be sent.

`deliveryMode` - The delivery mode to use.

`priority` - The priority for this message.

`timeToLive` - The message's lifetime (in milliseconds).

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicpublisher_close

Description

Since a provider may allocate some resources on behalf of a MessageProducer outside the JVM, clients should close them when they are not needed.

Function Signature

```
jint topicpublisher_close(FHandle mp);
```

Parameters

`mp` - The MessageProducer to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicpublisher_getDeliveryMode

Description

Gets the producer's default delivery mode.

Function Signature

```
jint topicpublisher_getDeliveryMode(FHandle mp, jint result);
```

Parameters

`mp` - The MessageProducer to operate on.

`result` - Contains the producer's default delivery mode.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicpublisher_getDisableMessageID

Description

Gets an indication of whether message IDs are disabled.

Function Signature

```
jint topicpublisher_getDisableMessageID(FHandle mp, jboolean re-
sult);
```

Parameters

`mp` - The MessageProducer to operate on.

`result` - Contains an indication of whether message IDs are disabled.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicpublisher_getDisableMessageTimestamp

Description

Gets an indication of whether message timestamps are disabled.

Function Signature

```
jint topicpublisher_getDisableMessageTimestamp(FHandle mp, jboolean
result);
```

Parameters

`mp` - The MessageProducer to operate on.

`result` - Contains an indication of whether message timestamps are disabled.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicpublisher_getPriority

Description

Gets the producer's default priority.

Function Signature

```
jint topicpublisher_getPriority(FHandle mp, jint result);
```

Parameters

`mp` - The MessageProducer to operate on.

`result` - Contains the message priority for this MessageProducer.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicpublisher_getTimeToLive

Description

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

Function Signature

```
jint topicpublisher_getTimeToLive(FHandle mp, jlong result);
```

Parameters

`mp` - The MessageProducer to operate on.

`result` - Contains the message time to live in milliseconds zero is unlimited.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicpublisher_setDeliveryMode

Description

Sets the producer's default delivery mode.

Function Signature

```
jint topicpublisher_setDeliveryMode(FHandle mp, jint deliveryMode);
```

Parameters

`mp` - The MessageProducer to operate on.

`deliveryMode` - The message delivery mode for this message producer. Legal values are DeliveryMode_NON_PERSISTENT or DeliveryMode_PERSISTENT.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicpublisher_setDisableMessageID

Description

Sets whether message IDs are disabled.

Function Signature

```
jint topicpublisher_setDisableMessageID(FHandle mp, jboolean value);
```

Parameters

`mp` - The MessageProducer to operate on.

`value` - Indicates if message IDs are to be disabled.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicpublisher_setDisableMessageTimestamp

Description

Sets whether message timestamps are disabled.

Function Signature

```
jint topicpublisher_setDisableMessageTimestamp(FHandle mp, jboolean value);
```

Parameters

`mp` - The MessageProducer to operate on.

`value` - Indicates if message timestamps are to be disabled.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicpublisher_setPriority

Description

Sets the producer's default priority.

Function Signature

```
jint topicpublisher_setPriority(FHandle mp, jint defaultPriority);
```

Parameters

`mp` - The MessageProducer to operate on.

`defaultPriority` - The message priority for this message producer. Priority must be a value between 0 and 9.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicpublisher_setTimeToLive

Description

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

Function Signature

```
jint topicpublisher_setTimeToLive(FHandle mp, jlong timeToLive);
```

Parameters

`mp` - The MessageProducer to operate on.

`timeToLive` - The message time to live in milliseconds zero is unlimited.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicpublisher_sendMessage

Description

Sends a message to a Destination for an unidentified message producer.

Function Signature

```
jint topicpublisher_sendMessage(FHandle mp, FHandle destination,
FHandle message);
```

Parameters

`mp` - The MessageProducer to operate on.

`destination` - The Destination to send the message to.

`message` - The message to be sent.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicpublisher_sendMessage_1

Description

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

Function Signature

```
jint topicpublisher_sendMessage_1FHandle mp, FHandle destination,
FHandle message, jint deliveryMode, jint priority, jlong timeToL-
ive);
```

Parameters

`mp` - The MessageProducer to operate on.

`destination` - The Destination to send the message to.

`message` - The message to be sent.

`deliveryMode` - The delivery mode to use.

`priority` - The priority for this message.

`timeToLive` - The message's lifetime (in milliseconds).

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicpublisher_sendMessage_2

Description

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

Function Signature

```
jint topicpublisher_sendMessage_2(FHandle mp, FHandle destination,
FHandle message);
```

Parameters

`mp` - The MessageProducer to operate on.

`message` - The message to be sent.

Returns

`MQ_SUCCESS` if successful, MQ_FAILURE if error.

topicpublisher_sendMessage_3

Description

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

Function Signature

```
jint topicpublisher_sendMessage_3(FHandle mp, FHandle message, jint deliveryMode, jint priority, jlong timeToLive);
```

Parameters

`mp` - The MessageProducer to operate on.

`message` - The message to be sent.

`deliveryMode` - The delivery mode to use.

`priority` - The priority for this message.

`timeToLive` - The message's lifetime (in milliseconds).

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

QueueRequestor

The QueueRequestor helper class simplifies making service requests.

createQueueRequestor

Description

Creates a new QueueRequestor object.

Function Signature

```
jint createQueueRequestor(FHandleFunction Signature qr, FHandle qsession, FHandle queue);
```

Parameters

`qr` - Contains the new QueueRequestor object.

`qsession` - The queue session the queue belongs to.

`queue` - The queue to perform the request/reply call on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuerequestor_close

Description

Since a provider may allocate some resources on behalf of a QueueRequestor outside the JVM, clients should close them when they are not needed.

Function Signature

```
jint queuerequestor_close(FHandle qr);
```

Parameters

`qr` - The QueueRequestor to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuerequestor_request

Description

Send a request and wait for a reply.

Function Signature

```
jint queuerequestor_request(FHandle qr, FHandle message, FHandle result);
```

Parameters

`qr` - The QueueRequestor to operate on.

`message` - The Message object to send.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

MessageConsumer

MessageConsumer object is used to receive messages from a destination. A MessageConsumer object is created by passing a Destination object to a message-consumer creation method supplied by a session. A message consumer can be created with a message selector. A message selector allows the client to restrict the messages delivered to the message consumer to those that match the selector.

messageconsumer_close

Description

Since a provider may allocate some resources on behalf of a MessageConsumer outside the JVM, clients should close them when they are not needed.

Function Signature

```
jint messageconsumer_close(FHandle mc);
```

Parameters

`mc` - The message consumer to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

messageconsumer_getMessageSelector

Description

Gets this message consumer's message selector expression.

Function Signature

```
jint messageconsumer_getMessageSelector(FHandle mc, const char *result);
```

Parameters

`mc` - The message consumer to operate on.

`result` - Contains this consumer's message selector expression.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

messageconsumer_receive

Description

Receives the next message produced for this message consumer.

Function Signature

```
jint messageconsumer_receive(FHandle mc, FHandle result);
```

Parameters

`mc` - The message consumer to operate on.

`result` - Contains the next Message produced for this message consumer.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

messageconsumer_receive_1

Description

Receives the next message that arrives within the specified timeout interval.

Function Signature

```
jint messageconsumer_receive_1(FHandle mc, jlong timeout, FHandle
result);
```

Parameters

`mc` - The message consumer to operate on.

`timeout` - The timeout value (in milliseconds).

`result` - Contains the next Message produced for this message consumer.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

messageconsumer_receiveNoWait

Description

Receives the next message if one is immediately available.

Function Signature

```
jint messageconsumer_receiveNoWait(FHandle mc, FHandle result);
```

Parameters

`mc` - The message consumer to operate on.

`result` - Contains the next Message produced for this message consumer.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

messageconsumer_setMessageListener

Description

Sets the message consumer's MessageListener.

Function Signature

```
jint messageconsumer_setMessageListener(FHandle mc, OnMessageMethod
messageListener);
```

Parameters

`mc` - The message consumer to operate on.

`messageListener` - The callback function to invoke when a message arrives.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

messageconsumer_setMessageListener_1

Description

Sets the message consumer's MessageListener.

Function Signature

```
jint messageconsumer_setMessageListener_1(FHandle mc, OnMes-
sageMethodWithParam messageListener, void param);
```

Parameters

`mc` - the message consumer to operate on.

`messageListener` - The callback function to invoke when a message arrives.

`param` - The parameter to be passed to the callback function.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

messageconsumer_getDestination

Description

Gets the Destination associated with this MessageConsumer.

Function Signature

```
jint messageconsumer_getDestination(FHandle mc, FHandle result);
```

Parameters

`mc` - The message consumer to operate on.

`result` - Contains the Destination object for this MessageConsumer.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

messageconsumer_getNoLocal

Description

Gets the NoLocal attribute for this MessageConsumer.

Function Signature

```
jint messageconsumer_getNoLocal(FHandle mc, jboolean result);
```

Parameters

`mc` - The message consumer to operate on.

`result` - Contains TRUE if locally published messages are being inhibited.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

QueueReceiver

QueueReceiver object is used to receive messages that have been delivered to a queue.

queuereceiver_getQueue

Description

Gets the queue associated with this queue receiver.

Function Signature

```
jint queuereceiver_getQueue(FHandle mc, FHandle result);
```

Parameters

`mc` - The message consumer to operate on.

`result` - Contains the Queue object associated with this message consumer.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuereceiver_close

Description

Since a provider may allocate some resources on behalf of a MessageConsumer outside the JVM, clients should close them when they are not needed.

Function Signature

```
jint queuereceiver_close(FHandle mc);
```

Parameters

`mc` - The message consumer to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

messageconsumer_getMessageSelector

Description

Gets this message consumer's message selector expression.

Function Signature

```
jint messageconsumer_getMessageSelector(FHandle mc, const char
*result);
```

Parameters

`mc` - The message consumer to operate on.

`result` - Contains this consumer's message selector expression.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuereceiver_receive

Description

Receives the next message produced for this message consumer.

Function Signature

```
jint queuereceiver_receive(FHandle mc, FHandle result);
```

Parameters

`mc` - The message consumer to operate on.

`result` - Contains the next Message produced for this message consumer.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuereceiver_receive_1

Description

Receives the next message that arrives within the specified timeout interval.

Function Signature

```
jint queuereceiver_receive_1(FHandle mc, jlong timeout, FHandle
result);
```

Parameters

`mc` - The message consumer to operate on.

`timeout` - The timeout value (in milliseconds).

`result` - Contains the mext Message produced for this message consumer.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuereceiver_receiveNoWait

Description

Receives the next message if one is immediately available.

Function Signature

```
jint queuereceiver_receiveNoWait(FHandle mc, FHandle result);
```

Parameters

`mc` - The message consumer to operate on.

`result` - Contains the mext Message produced for this message consumer.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuereceiver_setMessageListener

Description

Sets the message consumer's MessageListener.

Function Signature

```
jint queuereceiver_setMessageListener(FHandle mc, OnMessageMethod
messageListener);
```

Parameters

`mc` - The message consumer to operate on.

`messageListener` - The callback function to invoke when a message arrives.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuereceiver_setMessageListener_1

Description

Sets the message consumer's MessageListener.

Function Signature

```
jint queuereceiver_setMessageListener_1(FHandle mc, OnMes-
sageMethodWithParam messageListener, void param);
```

Parameters

`mc` - The message consumer to operate on.

`messageListener` - The callback function to invoke when a message arrives.

`param` - The parameter to be passed to the callback function.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuereceiver_getDestination

Description

Gets the Destination associated with this MessageConsumer.

Function Signature

```
jint queuereceiver_getDestination(FHandle mc, FHandle result);
```

Parameters

`mc` - The message consumer to operate on.

`result` -Contains the Destination object for this MessageConsumer.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

queuereceiver_getNoLocal

Description

Gets the NoLocal attribute for this MessageConsumer.

Function Signature

```
jint queuereceiver_getNoLocal(FHandle mc, jboolean result);
```

Parameters

`mc` - The message consumer to operate on.

`result` - Contains TRUE if locally published messages are being inhibited.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

TopicSubscriber

TopicSubscriber object is used to receive messages that have been published to a topic. A TopicSubscriber object is the publish/subscribe form of a message consumer. It allows the creation of multiple TopicSubscriber objects per topic.

topicsubscriber_getNoLocal

Description

Gets the NoLocal attribute for this TopicSubscriber.

Function Signature

```
jint topicsubscriber_getNoLocal(FHandle mc, jboolean result);
```

Parameters

`mc` - The message consumer to operate on.

`result` - Contains the noLocal attribute.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsubscriber_getTopic

Description

Gets the topic associated with this subscriber.

Function Signature

```
jint topicsubscriber_getTopic(FHandle mc, FHandle result);
```

Parameters

`mc` - The message consumer to operate on.

`result` - Contains the Topic object associated with this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsubscriber_close

Description

Since a provider may allocate some resources on behalf of a MessageConsumer outside the JVM, clients should close them when they are not needed.

Function Signature

```
jint topicsubscriber_close(FHandle mc);
```

Parameters

`mc` - The message consumer to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsubscriber_getMessageSelector

Description

Gets this message consumer's message selector expression.

Function Signature

```
jint topicsubscriber_getMessageSelector(FHandle mc, const char *re-
sult);
```

Parameters

`mc` - The message consumer to operate on.

`result` - Contains this consumer's message selector expression.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsubscriber_receive

Description

Receives the next message produced for this message consumer.

Function Signature

```
jint topicsubscriber_receive(FHandle mc, FHandle result);
```

Parameters

`mc` - The message consumer to operate on.

`result` - Contains the next Message produced for this message consumer.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsubscriber_receive_1

Description

Receives the next message that arrives within the specified timeout interval.

Function Signature

```
jint topicsubscriber_receive_1(FHandle mc, jlong timeout, FHandle
result);
```

Parameters

`mc` - The message consumer to operate on.

`timeout` - The timeout value (in milliseconds).

`result` - Contains the next Message produced for this message consumer.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsubscriber_receiveNoWait

Description

Receives the next message if one is immediately available.

Function Signature

```
jint topicsubscriber_receiveNoWait(FHandle mc, jlong timeout, FHan-
dle result);
```

Parameters

`mc` - The message consumer to operate on.

`result` - Contains the next Message produced for this message consumer.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsubscriber_setMessageListener

Description

Sets the message consumer's MessageListener.

Function Signature

```
jint topicsubscriber_setMessageListener(FHandle mc, OnMessageMethod
messageListener);
```

Parameters

`mc` - The message consumer to operate on.

`messageListener` - The callback function to invoke when a message arrives.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsubscriber_setMessageListener_1

Description

Sets the message consumer's MessageListener.

Function Signature

```
jint topicsubscriber_setMessageListener_1(FHandle mc, OnMes-
sageMethodWithParam messageListener, void param);
```

Parameters

`mc` - The message consumer to operate on.

`messageListener` - The callback function to invoke when a message arrives.

`param` - The parameter to be passed to the callback function.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

topicsubscriber_getDestination

Description

Gets the Destination associated with this MessageConsumer.

Function Signature

```
jint topicsubscriber_getDestination(FHandle mc, FHandle result);
```

Parameters

`mc` - The message consumer to operate on.

`result` - Contains the Destination object for this MessageConsumer.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

TemporaryQueue

A TemporaryQueue object is a unique Queue object created for the duration of a Connection. It is a system-defined queue and can be consumed only by the Connection that created it.

A TemporaryQueue object can be created at either the Session or QueueSession level. Creating it at the Session level allows to the TemporaryQueue to participate in transactions with objects from the Pub/Sub domain.

temporaryqueue_delete

Description

Deletes this temporary queue.

Function Signature

```
jint temporaryqueue_delete(FHandle tempQueue);
```

Parameters

`tempQueue` - The temporary queue to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

TemporaryTopic

A TemporaryTopic object is a unique Topic object created for the duration of a Connection. It is a system-defined topic that can be consumed only by the Connection that created it. A TemporaryTopic object can be created either at the Session or TopicSession level.

temporarytopic_delete

Description

Delete this temporary topic.

Function Signature

```
jint temporarytopic_delete(FHandle tempTopic);
```

Parameters

`tempTopic` - The temporary topic to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

Message

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

message_acknowledge

Description

Acknowledge this and all previous messages received.

Function Signature

```
jint message_acknowledge(FHandle message);
```

Parameters

`message` - The message to operate on.

message_clearBody

Description

Clear out the message body.

Function Signature

```
jint message_clearBody(FHandle message);
```

Parameters

`message` - The message to operate on.

message_clearProperties

Description

Clear a message's properties.

Function Signature

```
jint message_clearProperties(FHandle message);
```

Parameters

`message` - The message to operate on.

message_getBooleanProperty

Description

Return the boolean property value with the given name.

Function Signature

```
jint message_getBooleanProperty(FHandle message, const charFunction
Signature name, jboolean property);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`property` - Contains the value of the property after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_getByteProperty

Description

Return the byte property value with the given name.

Function Signature

```
jint message_getByteProperty(FHandle message, const char name,
jbyte property);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`property` - Contains the value of the property after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_getDoubleProperty

Description

Return the double property value with the given name.

Function Signature

```
jint message_getDoubleProperty(FHandle message, const char name,
jdouble property);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`property` - Contains the value of the property after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_getFloatProperty

Description

Return the float property value with the given name.

Function Signature

```
jint message_getFloatProperty(FHandle message, const char name,
jfloat property);
```

Parameters

`message` - The message to operate on.

`name` -The name of the property.

`property` - Contains the value of the property after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_getIntProperty

Description

Return the integer property value with the given name.

Function Signature

```
jint message_getIntProperty(FHandle message, const char name, jint
property);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`property` - Contains the value of the property after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_getJMSCorrelationID

Description

Gets the correlation ID for the message.

Function Signature

```
jint message_getJMSCorrelationID(FHandle message, const char *cor-
relationID);
```

Parameters

`message` - The message to operate on.

`correlationID` - The correlation ID of this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_getJMSCorrelationIDAsBytes

Description

Gets the correlation ID as an array of bytes for the message.

Function Signature

```
jint message_getJMSCorrelationIDAsBytes(FHandle message, jbyte
*correlationID);
```

Parameters

`message` - The message to operate on.

`correlationID` - The correlation ID of this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_getJMSDeliveryMode

Description

Gets the delivery mode for this message.

Function Signature

```
jint message_getJMSDeliveryMode(FHandle message, jint delivery-
Mode);
```

Parameters

`message` - The message to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_getJMSDestination

Description

Gets the destination for this message.

Function Signature

```
jint message_getJMSDestination(FHandle message, FHandle destina-
tion);
```

Parameters

`message` - The message to operate on.

`destination` - The destination of this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_getJMSExpiration

Description

Gets the message's expiration value.

Function Signature

```
jint message_getJMSExpiration(FHandle message, jlong expiration);
```

Parameters

`message` - The message to operate on.

`expiration` - The time the message expires.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_getJMSMessageID

Description

Gets the message ID.

Function Signature

```
jint message_getJMSMessageID(FHandle message, const char *mes-
sageID);
```

Parameters

`message` - The message to operate on.

`messageID` - The message ID.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_getJMSPriority

Description

Gets the message priority.

Function Signature

```
jint message_getJMSPriority(FHandle message, jint priority);
```

Parameters

`message` - The message to operate on.

`priority` - The priority for this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_getJMSRedelivered

Description

Gets an indication of whether this message is being redelivered.

Function Signature

```
jint message_getJMSRedelivered(FHandle message, jboolean redeliv-
ered);
```

Parameters

`message` - The message to operate on.

`redelivered` - TRUE if this message is being redelivered.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_getJMSReplyTo

Description

Gets where a reply to this message should be sent.

Function Signature

```
jint message_getJMSReplyTo(FHandle message, FHandle destination);
```

Parameters

`message` - The message to operate on.

`destination` - Where a reply to this message should be sent.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_getJMSTimestamp

Description

Gets the message timestamp.

Function Signature

```
jint message_getJMSTimestamp(FHandle message, jlong timestamp);
```

Parameters

`message` - The message to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_getJMSType

Description

Gets the message type.

Function Signature

```
jint message_getJMSType(FHandle message, const charFunction Signa-
ture *type);
```

Parameters

`message` - The message to operate on.

`type` - the type of the message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_getLongProperty

Description

Return the long property value with the given name.

Function Signature

```
jint message_getLongProperty(FHandle message, const char name, jlong
property);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`property` - Contains the value of the property after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_getPropertyNames

Description

Return an Enumeration of all the property names.

Function Signature

```
jint message_getPropertyNames(FHandle message, FHandle enumera-
tion);
```

Parameters

`message` - The message to operate on.

`enumeration` - An enumeration of property names.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_getShortProperty

Description

Return the short property value with the given name.

Function Signature

```
jint message_getShortProperty(FHandle message, const char name,
jshort property);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`property` - The value of the property.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_getStringProperty

Description

Return the String property value with the given name.

Function Signature

```
jint message_getStringProperty(FHandle message, const char name,
const char *property);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`property` - The value of the property.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_propertyExists

Description

Check if a property value exists.

Function Signature

```
jint message_propertyExists(FHandle message, const char name,
jboolean exists);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`exists` - Indicates whether the property exists or not.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_setBooleanProperty

Description

Sets a boolean property value with the given name, into the Message.

Function Signature

```
jint message_setBooleanProperty(FHandle message, const char name,
jboolean value);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`value` - The property value with the given name.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_setByteProperty

Description

Sets a byte property value with the given name, into the Message.

Function Signature

```
jint message_setByteProperty(FHandle message, const charname, jbyte
value);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`value` - The property value with the given name.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_setDoubleProperty

Description

Sets a double property value with the given name, into the Message.

Function Signature

```
jint message_setDoubleProperty(FHandle message, const char name,
jdouble value);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`value` - The property value with the given name.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_setFloatProperty

Description

Sets a float property value with the given name, into the Message.

Function Signature

```
jint message_setFloatProperty(FHandle message, const char name,
jfloat value);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`value` - The property value with the given name.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_setIntPropertye

Description

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

Function Signature

```
jint message_setIntProperty(FHandle message, const char name, jint value);
```

Parameters

`message` - Tthe message to operate on.

`name` - The name of the property.

`value` - The property value with the given name.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_setJMSCorrelationID

Description

Sets the correlation ID for the message.

Function Signature

```
jint message_setJMSCorrelationID(FHandle message, const char correlationID);
```

Parameters

`message` - The message to operate on.

`correlationID` - The correlation ID for the message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_setJMSCorrelationIDAsBytes

Description

Sets the correlation ID as an array of bytes for the message.

Function Signature

```
jint message_setJMSCorrelationIDAsBytes(FHandle message, jbyte cor-
relationID, jint length);
```

Parameters

`message` - The message to operate on.

`correlationID` - The correlation ID value as an array of bytes.

`length` - The length of the array.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_setJMSDeliveryMode

Description

Sets the delivery mode for this message.

Function Signature

```
jint message_setJMSDeliveryMode(FHandle message, jint delivery-
Mode);
```

Parameters

`message` - The message to operate on.

`deliveryMode` - The delivery mode for this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_setJMSDestination

Description

Sets the destination for this message.

Function Signature

```
jint message_setJMSDestination(FHandle message, FHandle destina-
tion);
```

Parameters

`message` - The message to operate on.

`destination` - the destination for this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_setJMSExpiration

Description

Sets the message's expiration value.

Function Signature

```
jint message_setJMSExpiration(FHandle message, jlong expiration);
```

Parameters

`message` - The message to operate on.
`expiration` - The message's expiration time.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_setJMSMessageID

Description

Sets the message ID.

Function Signature

```
jint message_setJMSMessageID(FHandle message, const char id);
```

Parameters

`message` - The message to operate on.
`id` - The message ID.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_setJMSPriority

Description

Sets the priority for this message.

Function Signature

```
jint message_setJMSPriority(FHandle message, jint priority);
```

Parameters

`message` - The message to operate on.

`priority` - The priority for this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_setJMSRedelivered

Description

Sets to indicate whether this message is being redelivered.

Function Signature

```
jint message_setJMSRedelivered(FHandle message, jboolean redeliv-
ered);
```

Parameters

`message` - The message to operate on.

`redelivered` - An indication of whether this message is being redelivered.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_setJMSReplyTo

Description

Sets where a reply to this message should be sent.

Function Signature

```
jint message_setJMSReplyTo(FHandle message, FHandle replyTo);
```

Parameters

`message` - The message to operate on.

`replyTo` - The destination to reply to.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_setJMSTimestamp

Description

Sets the message timestamp.

Function Signature

```
jint message_setJMSTimestamp(FHandle message, jlong timestamp);
```

Parameters

`message` - The message to operate on.
`timestamp` - The timestamp for this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_setJMSType

Description

Sets the message type.

Function Signature

```
jint message_setJMSType(FHandle message, const char type);
```

Parameters

`message` - The message to operate on.
`type` - The class of message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_setLongProperty

Description

Sets a long property value with the given name, into the Message.

Function Signature

```
jint message_setLongProperty(FHandle message, const char name,
jlong value);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`value` - The property value with the given name.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_setShortProperty

Description

Sets a short property value with the given name, into the Message.

Function Signature

```
jint message_setShortProperty(FHandle message, const char name,
jshort value);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`value` - The property value with the given name.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

message_setStringProperty

Description

Sets a String property value with the given name, into the Message.

Function Signature

```
jint message_setStringProperty(FHandle message, const char name,
const char value);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`value` - The property value with the given name.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

TextMessage

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

textmessage_setText

Description

Sets the string containing this message's data

Function Signature

```
jint textmessage_setText(FHandle txtmsg, const char text);
```

Parameters

`txtmsg` - The text message to operate on.

`text` - The string containing this message's data.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_getText

Description

Gets the string containing this message's data

Function Signature

```
jint textmessage_getText(FHandle txtmsg, const char *text);
```

Parameters

`txtmsg` - The text message to operate on.

`text` - Contains this message's data after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_acknowledge

Description

Acknowledge this and all previous messages received.

Function Signature

```
jint textmessage_acknowledge(FHandle message);
```

Parameters

`message` - The message to operate on.

textmessage_clearBody

Description

Clear out the message body.

Function Signature

```
jint textmessage_clearBody(FHandle message);
```

Parameters

`message` - The message to operate on.

textmessage_clearProperties

Description

Clear a message's properties.

Function Signature

```
jint textmessage_clearProperties(FHandle message);
```

Parameters

`message` - The message to operate on.

textmessage_getBooleanProperty

Description

Return the boolean property value with the given name.

Function Signature

```
jint textmessage_getBooleanProperty(FHandle message, const char
name, jboolean property);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`property` - Contains the value of the property after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_getByteProperty

Description

Return the byte property value with the given name.

Function Signature

```
jint textmessage_getByteProperty(FHandle message, const char name,
jbyte property);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`property` - Contains the value of the property after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_getDoubleProperty

Description

Return the double property value with the given name.

Function Signature

```
jint textmessage_getDoubleProperty(FHandle message, const char
name, jdouble property);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`property` - Contains the value of the property after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_getFloatProperty

Description

Return the float property value with the given name.

Function Signature

```
jint textmessage_getFloatProperty(FHandle message, const char name,
jfloat property);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`property` - Contains the value of the property after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_getIntProperty

Description

Return the integer property value with the given name.

Function Signature

```
jint textmessage_getIntProperty(FHandle message, const char name,
jint property);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`property` - Contains the value of the property after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_getJMSCorrelationID

Description

Gets the correlation ID for the message.

Function Signature

```
jint textmessage_getJMSCorrelationID(FHandle message, const char
*correlationID);
```

Parameters

`message` - The message to operate on.

`correlationID` - The correlation ID of this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_getJMSCorrelationIDAsBytes

Description

Gets the correlation ID as an array of bytes for the message.

Function Signature

```
jint textmessage_getJMSCorrelationIDAsBytes(FHandle message, jbyte
*correlationID);
```

Parameters

`message` - The message to operate on.

`correlationID` - The correlation ID of this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_getJMSDeliveryMode

Description

Gets the delivery mode for this message.

Function Signature

```
jint textmessage_getJMSDeliveryMode(FHandle message, jint delivery-
Mode);
```

Parameters

`message` - The message to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_getJMSDestination

Description

Gets the destination for this message.

Function Signature

```
jint textmessage_getJMSDestination(FHandle message, FHandle desti-
nation);
```

Parameters

`message` - The message to operate on.

`destination` - The destination of this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_getJMSExpiration

Description

Gets the message's expiration value.

Function Signature

```
jint textmessage_getJMSExpiration(FHandle message, jlong expira-
tion);
```

Parameters

`message` - The message to operate on.

`expiration` - The time the message expires.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_getJMSMessageID

Description

Gets the message ID.

Function Signature

```
jint textmessage_getJMSMessageID(FHandle message, const char *mes-
sageID);
```

Parameters

`message` - The message to operate on.

`messageID` the message ID.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.s

textmessage_getJMSPriority

Description

Gets the message priority.

Function Signature

```
jint textmessage_getJMSPriority(FHandle message, jint priority);
```

Parameters

`message` -The message to operate on.

`priority` - The priority for this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_getJMSRedelivered

Description

Gets an indication of whether this message is being redelivered.

Function Signature

```
jint textmessage_getJMSRedelivered(FHandle message, jboolean rede-
livered);
```

Parameters

`message` - The message to operate on.

`redelivered` - TRUE if this message is being redelivered.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_getJMSReplyTo

Description

Gets where a reply to this message should be sent.

Function Signature

```
jint textmessage_getJMSReplyTo(FHandle message, FHandle destina-
tion);
```

Parameters

`message` - The message to operate on.

`destination` - Where a reply to this message should be sent.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_getJMSTimestamp

Description

Gets the message timestamp.

Function Signature

```
jint textmessage_getJMSTimestamp(FHandle message, jlong timestamp);
```

Parameters

`message` - The message to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_getJMSType

Description

Gets the message type.

Function Signature

```
jint textmessage_getJMSType(FHandle message, const char *type);
```

Parameters

`message` - The message to operate on.

`type` - The type of the message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_getLongProperty

Description

Return the long property value with the given name.

Function Signature

```
jint textmessage_getLongProperty(FHandle message, const char name,
jlong property);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`property` - Contains the value of the property after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_getPropertyNames

Description

Returns an Enumeration of all the property names.

Function Signature

```
jint textmessage_getPropertyNames(FHandle message, FHandle enumera-
tion);
```

Parameters

`message` - The message to operate on.

`enumeration` - An enumeration of property names.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_getShortProperty

Description

Return the short property value with the given name.

Function Signature

```
jint textmessage_getShortProperty(FHandle message, const char name,
jshort property);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`property` - The value of the property.

Returns

`MQ_SUCCESS` if successful, MQ_FAILURE if error.

textmessage_getStringProperty

Description

Return the String property value with the given name.

Function Signature

```
jint textmessage_getStringProperty(FHandle message, const char name,
const char *property);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`property` - The value of the property.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_propertyExists

Description

Check if a property value exists.

Function Signature

```
jint textmessage_propertyExists(FHandle message, const char name,
jboolean exists);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`exists` - Indicates whether the property exists or not.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_setBooleanProperty

Description

Sets a boolean property value with the given name, into the Message.

Function Signature

```
jint textmessage_setBooleanProperty(FHandle message, const char
name, jboolean value);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`value` - The property value with the given name.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_setByteProperty

Description

Sets a byte property value with the given name, into the Message.

Function Signature

```
jint textmessage_setByteProperty(FHandle message, const charname,
jbyte value);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`value` - The property value with the given name.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_setDoubleProperty

Description

Sets a double property value with the given name, into the Message.

Function Signature

```
jint textmessage_setDoublePropertyFHandle message, const char name,
jdouble value);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`value` - The property value with the given name.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_setFloatProperty

Description

Sets a float property value with the given name, into the Message.

Function Signature

```
jint textmessage_setFloatProperty(FHandle message, const char name,
jfloat value);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`value` - The property value with the given name.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_setIntProperty

Description

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

Function Signature

```
jint textmessage_setIntProperty(FHandle message, const char name,
jint value);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`value` - The property value with the given name.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_setJMSCorrelationID

Description

Sets the correlation ID for the message.

Function Signature

```
jint textmessage_setJMSCorrelationID(FHandle message, const char
correlationID);
```

Parameters

`message` - The message to operate on.

`correlationID` - The correlation ID for the message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_setJMSCorrelationIDAsBytes

Description

Sets the correlation ID as an array of bytes for the message.

Function Signature

```
jint textmessage_setJMSCorrelationIDAsBytes(FHandle message, jbyte
correlationID, jint length);
```

Parameters

`message` - The message to operate on.

`correlationID` - The correlation ID value as an array of bytes.

`length` the length of the array.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_setJMSDeliveryMode

Description

Sets the delivery mode for this message.

Function Signature

```
jint textmessage_setJMSDeliveryMode(FHandle message, jint delivery-
Mode);
```

Parameters

`message` - The message to operate on.

`deliveryMode` - The delivery mode for this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_setJMSDestination

Description

Sets the destination for this message.

Function Signature

```
jint textmessage_setJMSDestination(FHandle message, FHandle desti-
nation);
```

Parameters

`message` - The message to operate on.

`destination` - the destination for this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_setJMSExpiration

Description

Sets the message's expiration value.

Function Signature

```
jint textmessage_setJMSExpiration(FHandle message, jlong expira-
tion);
```

Parameters

`message` - The message to operate on.

`expiration` - The message's expiration time.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_setJMSMessageID

Description

Sets the message ID.

Function Signature

```
jint textmessage_setJMSMessageID(FHandle message, const char id);
```

Parameters

`message` - The message to operate on.

`id` - The message ID.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_setJMSPriority

Description

Sets the priority for this message.

Function Signature

```
jint textmessage_setJMSPriority(FHandle message, jint priority);
```

Parameters

`message` - The message to operate on.

`priority` - The priority for this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_setJMSRedelivered

Description

Sets to indicate whether this message is being redelivered.

Function Signature

```
jint textmessage_setJMSRedelivered(FHandle message, jboolean rede-
livered);
```

Parameters

`message` - The message to operate on.

`redelivered` - An indication of whether this message is being redelivered.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if errors.

textmessage_setJMSReplyTo

Description

Sets where a reply to this message should be sent.

Function Signature

```
jint textmessage_setJMSReplyTo(FHandle message, FHandle replyTo);
```

Parameters

`message` - The message to operate on.

`replyTo` - The destination to reply to.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_setJMSTimestamp

Description

Sets the message timestamp.

Function Signature

```
jint textmessage_setJMSTimestamp(FHandle message, jlong timestamp);
```

Parameters

`message` - The message to operate on.

`timestamp` - The timestamp for this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_setJMSType

Description

Sets the message type.

Function Signature

```
jint textmessage_setJMSType(FHandle message, const char type);
```

Parameters

`message` - The message to operate on.

`type` - The class of message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_setLongProperty

Description

Sets a long property value with the given name, into the Message.

Function Signature

```
jint textmessage_setLongProperty(FHandle message, const char name,
jlong value);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`value` - The property value with the given name.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_setShortProperty

Description

Sets a short property value with the given name, into the Message.

Function Signature

```
jint textmessage_setShortProperty(FHandle message, const char name,
jshort value);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`value` - The property value with the given name.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

textmessage_setStringProperty

Description

Sets a String property value with the given name, into the Message.

Function Signature

```
jint textmessage_setStringProperty(FHandle message, const char
name, const char value);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`value` - The property value with the given name.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

BytesMessage

A BytesMessage object is used to send a message containing a stream of uninterpreted bytes. It inherits from the Message interface and adds a bytes message body. The receiver of the message supplies the interpretation of the bytes. The BytesMessage methods are based largely on those found in java.io.DataInputStream and java.io.DataOutputStream.

bytesmessage_readBoolean

Description

Read a boolean from the bytes message stream.

Function Signature

```
jint bytesmessage_readBoolean(FHandle bm, jboolean value);
```

Parameters

`bm` - The bytes message to operate on.

`value` - Contains the value that is read.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_readByte

Description

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

Function Signature

```
jint bytesmessage_readByte(FHandle bm, jbyte value);
```

Parameters

`bm` - The bytes message to operate on.

`value` - Contains the value that is read.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_readBytes

Description

Read a byte array from the bytes message stream.

Function Signature

```
jint bytesmessage_readBytes(FHandle bm, jbyte *value, jint value-
Length, jint noOfBytes);
```

Parameters

`bm` - The bytes message to operate on.

`value` - The buffer into which the data is read.

`valueLength` - The length of the value array.

`noOfBytes` - The number of bytes read.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_readBytes_1

Description

Read a portion of the bytes message stream.

Function Signature

```
jint bytesmessage_readBytes_1(FHandle bm, jbyte *value, jint value-
Length, jint length, jint noOfBytes);
```

Parameters

`bm` - The bytes message to operate on.

`value` - Contains the value that is read.

`valueLength` - The length of the value array.

`length` - The number of bytes to read.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_readChar

Description

Read a Unicode character value from the bytes message stream.

Function Signature

```
jint bytesmessage_readChar(FHandle bm, char value);
```

Parameters

`bm` - The bytes message to operate on.

`value` - Contains the value that is read.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_readDouble

Description

Read a double from the bytes message stream.

Function Signature

```
jint bytesmessage_readDouble(FHandle bm, jdouble value);
```

Parameters

`bm` - The bytes message to operate on.

`value` - Contains the value that is read.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_readFloat

Description

Read a float from the bytes message stream.

Function Signature

```
jint bytesmessage_readFloat(FHandle bm, jfloat value);
```

Parameters

`bm` - The bytes message to operate on.

`value` - Contains the value that is read.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_readInt

Description

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

Function Signature

```
jint bytesmessage_readInt(FHandle bm, jint value);
```

Parameters

`bm` - The bytes message to operate on.

`value` - Contains the value that is read.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_readLong

Description

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

Function Signature

```
jint bytesmessage_readLong(FHandle bm, jlong value);
```

Parameters

`bm` - The bytes message to operate on.

`value` - Contains the value that is read.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_readShort

Function Signature

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

Function Signature

```
jint bytesmessage_readShort(FHandle bm, jshort value);
```

Parameters

`bm` - The bytes message to operate on.

`value` - Contains the value that is read.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_readUnsignedByte

Description

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

Function Signature

```
jint bytesmessage_readUnsignedByte(FHandle bm, jint value);
```

Parameters

`bm` - The bytes message to operate on.

`value` - Contains the value that is read.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_readUnsignedShort

Description

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

Function Signature

```
jint bytesmessage_readUnsignedShort(FHandle bm, jint value);
```

Parameters

`bm` - The bytes message to operate on.

`value` - Contains the value that is read.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_readUTF

Description

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

Function Signature

```
jint bytesmessage_readUTF(FHandle bm, const char value);
```

Parameters

`bm` - The bytes message to operate on.

`value` - Contains the value that is read.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_reset

Description

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

Function Signature

```
jint bytesmessage_reset(FHandle bm);
```

Parameters

`bm` - The bytes message to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_writeBoolean

Description

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

Function Signature

```
jint bytesmessage_writeBoolean(FHandle bm, jboolean value);
```

Parameters

`bm` - The bytes message to operate on.

`value` - The value to be written

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_writeByte

Description

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

Function Signature

```
jint bytesmessage_writeByte(FHandle bm, jbyte value);
```

Parameters

`bm` - The bytes message to operate on.

`value` - The value to be written.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_writeBytes

Description

Writes a byte array to the bytes message stream.

Function Signature

```
jint bytesmessage_writeBytes(FHandle bm, jbyte value, jint value-
Length);
```

Parameters

`bm` - The bytes message to operate on.

`value` - The value to be written.

`valueLength` - The length of the value array.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_writeBytes_1

Description

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

Function Signature

```
jint bytesmessage_writeBytes_1(FHandle bm, jbyte value, jint value-
Length, jint offset, jint length);
```

Parameters

`bm` - The bytes message to operate on.

`value` - The value to be written.

`valueLength` - The length of the value array.

`offset` - The initial offset within the byte array.

`length` - The number of bytes to use.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_writeChar

Description

Function Signature

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

Function Signature

```
jint bytesmessage_writeChar(FHandle bm, char value);
```

Parameters

`bm` - The bytes message to operate on.

`value` - the value to be written.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_writeDouble

Description

Convert the double argument to a long, and then writes that long value to the bytes message stream as an 8-byte quantity, high byte first. Write a char to the bytes message stream as a 2-byte value, high byte first.

Function Signature

```
jint bytesmessage_writeDouble(FHandle bm, jdouble value);
```

Parameters

`bm` - The bytes message to operate on.

`value` - The value to be written.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_writeFloat

Description

Convert the float argument to an int, and then writes that int value to the bytes message stream as a 4-byte quantity, high byte first.

Function Signature

```
jint bytesmessage_writeFloat(FHandle bm, jfloat value);
```

Parameters

`bm` - The bytes message to operate on.

`value` - The value to be written.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_writeInt

Description

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

Function Signature

```
jint bytesmessage_writeInt(FHandle bm, jint value);
```

Parameters

`bm` - The bytes message to operate on.

`value` - The value to be written.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_writeLong

Description

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

Function Signature

```
jint bytesmessage_writeLong(FHandle bm, jlong value);
```

Parameters

`bm` - The bytes message to operate on.

`value` - The value to be written.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_writeShort

Description

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

Function Signature

```
jint bytesmessage_writeShort(FHandle bm, jshort value);
```

Parameters

`bm` - The bytes message to operate on.

`value` - The value to be written.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_writeUTF

Description

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

Function Signature

```
jint bytesmessage_writeUTF(FHandle bm, const char value);
```

Parameters

`bm` - The bytes message to operate on.

`value` - The value to be written.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_acknowledge

Description

Acknowledge this and all previous messages received.

Function Signature

```
jint bytesmessage_acknowledge(FHandle message);
```

Parameters

`message` - The message to operate on.

bytesmessage_clearBody

Description

Clear out the message body.

Function Signature

```
jint bytesmessage_clearBody(FHandle message);
```

Parameters

`message` - The message to operate on.

bytesmessage_clearProperties

Description

Clear a message's properties.

Function Signature

```
jint bytesmessage_clearProperties(FHandle message);
```

Parameters

`message` - The message to operate on.

bytesmessage_getBooleanProperty

Description

Return the boolean property value with the given name.

Function Signature

```
jint bytesmessage_getBooleanProperty(FHandle message, const char
name, jboolean property);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`property` - Contains the value of the property after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_getByteProperty

Description

Return the byte property value with the given name.

Function Signature

```
jint bytesmessage_getByteProperty(FHandle message, const char name,
jbyte property);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`property` - Contains the value of the property after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_getDoubleProperty

Description

Return the double property value with the given name.

Function Signature

```
jint bytesmessage_getDoubleProperty(FHandle message, const char
name, jdouble property);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`property` - Contains the value of the property after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_getFloatProperty

Description

Return the float property value with the given name.

Function Signature

```
jint bytesmessage_getFloatProperty(FHandle message, const char name,
jfloat property);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`property` - Contains the value of the property after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_getIntProperty

Description

Return the integer property value with the given name.

Function Signature

```
jint bytesmessage_getIntProperty(FHandle message, const char name,
jint property);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`property` - Contains the value of the property after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_getJMSCorrelationID

Description

Gets the correlation ID for the message.

Function Signature

```
jint bytesmessage_getJMSCorrelationID(FHandle message, const char
*correlationID);
```

Parameters

`message` - The message to operate on.

`correlationID` - The correlation ID of this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_getJMSCorrelationIDAsBytes

Description

Gets the correlation ID as an array of bytes for the message.

Function Signature

```
jint bytesmessage_getJMSCorrelationIDAsBytes(FHandle message, jbyte
*correlationID);
```

Parameters

`message` - The message to operate on.

`correlationID` - The correlation ID of this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_getJMSDeliveryMode

Description

Gets the delivery mode for this message.

Function Signature

```
jint bytesmessage_getJMSDeliveryMode(FHandle message, jint deliv-
eryMode);
```

Parameters

`message` - The message to operate on.

Returns

`MQ_SUCCESS` if successful, MQ_FAILURE if error.

bytesmessage_getJMSDestination

Description

Gets the destination for this message.

Function Signature

```
jint bytesmessage_getJMSDestination(FHandle message, FHandle destination);
```

Parameters

`message` - The message to operate on.

`destination` - The destination of this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_getJMSExpiration

Description

Gets the message's expiration value.

Function Signature

```
jint bytesmessage_getJMSExpiration(FHandle message, jlong expiration);
```

Parameters

`message` - The message to operate on.

`expiration` - The time the message expires.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_getJMSMessageID

Description

Gets the message ID.

Function Signature

```
jint bytesmessage_getJMSMessageID(FHandle message, const char *messageID);
```

Parameters

`message` - The message to operate on.

`messageID` - The message ID.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_getJMSPriority

Description

Gets the message priority.

Function Signature

```
jint bytesmessage_getJMSPriority(FHandle message, jint priority);
```

Parameters

`message` - The message to operate on.

`priority` - The priority for this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_getJMSRedelivered

Description

Gets an indication of whether this message is being redelivered.

Function Signature

```
jint bytesmessage_getJMSRedelivered(FHandle message, jboolean rede-
livered);
```

Parameters

`message` - The message to operate on.

`redelivered` - TRUE if this message is being redelivered.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_getJMSReplyTo

Description

Gets where a reply to this message should be sent.

Function Signature

```
jint bytesmessage_getJMSReplyTo(FHandle message, FHandle destina-
tion);
```

Parameters

`message` - The message to operate on.

`destination` - Where a reply to this message should be sent.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_getJMSTimestamp

Description

Gets the message timestamp.

Function Signature

```
jint bytesmessage_getJMSTimestamp(FHandle message, jlong times-
tamp);
```

Parameters

`message` - The message to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_getJMSType

Description

Gets the message type.

Function Signature

```
jint bytesmessage_getJMSType(FHandle message, const charFunction
Signature *type);
```

Parameters

`message` - The message to operate on.

`type` - The type of the message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_getLongProperty

Description

Return the long property value with the given name.

Function Signature

```
jint bytesmessage_getLongProperty(message, name, property)
message_getLongProperty(message, name, property);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`property` - Contains the value of the property after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_getPropertyNames

Description

Return an Enumeration of all the property names.

Function Signature

```
jint bytesmessage_getPropertyNames(FHandle message, FHandle enumer-
ation);
```

Parameters

`message` - The message to operate on.

`enumeration` - An enumeration of property names.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_getShortProperty

Description

Return the short property value with the given name.

Function Signature

```
jint bytesmessage_getShortProperty(FHandle message, const char name,
jshort property);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`property` - The value of the property.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_getStringProperty

Description

Return the String property value with the given name.

Function Signature

```
jint bytesmessage_getStringProperty(FHandle message, const char
name, const char *property);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`property` - The value of the property.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_propertyExists

Description

Check if a property value exists.

Function Signature

```
jint bytesmessage_propertyExists(FHandle message, const char name,
jboolean exists);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`exists` - Indicates whether the property exists or not.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_setBooleanProperty

Description

Sets a boolean property value with the given name, into the Message.

Function Signature

```
jint bytesmessage_setBooleanProperty(FHandle message, const char
name, jboolean value);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`value` - The property value with the given name.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_setByteProperty

Description

Sets a byte property value with the given name, into the Message.

Function Signature

```
jint bytesmessage_setByteProperty(FHandle message, const charname,
jbyte value);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`value` - The property value with the given name.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_setDoubleProperty

Description

Sets a double property value with the given name, into the Message.

Function Signature

```
jint bytesmessage_setDoublePropertyFHandle message, const char name,
jdouble value);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`value` - The property value with the given name.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_setFloatProperty

Description

Sets a float property value with the given name, into the Message.

Function Signature

```
jint bytesmessage_setFloatProperty(FHandle message, const char name,
jfloat value);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`value` - The property value with the given name.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_setIntProperty

Description

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

Function Signature

```
jint bytesmessage_setIntProperty(FHandle message, const char name,
jint value);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`value` - The property value with the given name.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_setJMSCorrelationID

Description

Sets the correlation ID for the message.

Function Signature

```
jint bytesmessage_setJMSCorrelationIDFHandle message, const char
correlationID);
```

Parameters

`message` - The message to operate on.

`correlationID` - The correlation ID for the message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_setJMSCorrelationIDAsBytes

Description

Sets the correlation ID as an array of bytes for the message.

Function Signature

```
jint bytesmessage_setJMSCorrelationIDAsBytes(FHandle message, jbyte correlationID, jint length);
```

Parameters

`message` - The message to operate on.

`correlationID` - The correlation ID value as an array of bytes.

`length` -The length of the array.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_setJMSDeliveryMode

Description

Sets the delivery mode for this message.

Function Signature

```
jint bytesmessage_setJMSDeliveryMode(FHandle message, jint deliveryMode);
```

Parameters

`message` - The message to operate on.

`deliveryMode` - The delivery mode for this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_setJMSDestination

Description

Sets the destination for this message.

Function Signature

```
jint bytesmessage_setJMSDestination(FHandle message, FHandle destination);
```

Parameters

`message` - The message to operate on.

`destination` - The destination for this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_setJMSExpiration

Description

Sets the message's expiration value.

Function Signature

```
jint bytesmessage_setJMSExpiration(FHandle message, jlong expiration);
```

Parameters

`message` - The message to operate on.

`expiration` - The message's expiration time.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_setJMSMessageID

Description

Sets the message ID.

Function Signature

```
jint bytesmessage_setJMSMessageID(FHandle message, const char id);
```

Parameters

`message` - The message to operate on.

`id` - The message ID.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_setJMSPriority

Description

Sets the priority for this message.

Function Signature

```
jint bytesmessage_setJMSPriority(FHandle message, jint priority);
```

Parameters

`message` - The message to operate on.

`priority` - The priority for this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_setJMSRedelivered

Description

Sets to indicate whether this message is being redelivered.

Function Signature

```
jint bytesmessage_setJMSRedelivered(FHandle message, jboolean rede-
livered);
```

Parameters

`message` - The message to operate on.

`redelivered` - An indication of whether this message is being redelivered.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_setJMSReplyTo

Description

Sets where a reply to this message should be sent.

Function Signature

```
jint bytesmessage_setJMSReplyTo(FHandle message, FHandle replyTo);
```

Parameters

`message` - The message to operate on.

`replyTo` - The destination to reply to.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_setJMSTimestamp

Description

Sets the message timestamp.

Function Signature

```
jint bytesmessage_setJMSTimestamp(FHandle message, jlong times-
tamp);
```

Parameters

`message` - The message to operate on.

`timestamp` - The timestamp for this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_setJMSType

Description

Sets the message type.

Function Signature

```
jint bytesmessage_setJMSType(FHandle message, const char type);
```

Parameters

`message` - The message to operate on.

`type` - The class of message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_setLongProperty

Description

Sets a long property value with the given name, into the Message.

Function Signature

```
jint bytesmessage_setLongProperty(FHandle message, const char name,
jlong value);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`value` - The property value with the given name.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_setShortProperty

Description

Sets a short property value with the given name, into the Message.

Function Signature

```
jint bytesmessage_setShortProperty(FHandle message, const char name,
jshort value);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`value` - The property value with the given name.

Returns he

MQ_SUCCESS if successful, MQ_FAILURE if error.

bytesmessage_setStringProperty

Description

Sets a String property value with the given name, into the Message.

Function Signature

```
jint bytesmessage_setStringProperty(FHandle message, const char
name, const char value);
```

Parameters

`message` - The message to operate on.

`name` - The name of the property.

`value` - The property value with the given name.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

XAConnectionFactory

The XAConnectionFactory interface is a base interface for the XAQueueConnectionFactory and XATopicConnectionFactory interfaces.

xacf_createXAConnection

Description

Creates an XAConnection with default user identity.

Function Signature

```
jint xacf_createXAConnection(FHandle xacf, FHandle result);
```

Parameters

`xacf` - The XAConnectionFactory to operate on.

`result` - A newly created XAConnection.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xacf_createXAConnection_1

Description

Creates an XAConnection with specific user identity.

Function Signature

```
jint xacf_createXAConnection_1(FHandle xacf, const char username,
const char password, FHandle result);
```

Parameters

`xacf` - The XAConnectionFactory to operate on.

`username` - The caller's username.

`password` - The caller's password.

`result` - A newly created XAConnection.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

XAQueueConnectionFactory

An XAQueueConnectionFactory provides the same create options as a QueueConnectionFactory. This interface is for use by JMS providers to support transactional environments.

xaqcf_createXAQueueConnection

Description

Creates an XA queue connection with default user identity.

Function Signature

```
jint xaqcf_createXAQueueConnection(FHandle xaqcf, FHandle xaqcon-
nection);
```

Parameters

`xaqcf` - The XA queue connection factory to operate on.

`xaqconnection` - A newly created XA queue connection.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcf_createXAQueueConnection_1

Description

Creates an XA queue connection with specific user identity.

Function Signature

```
jint xaqcf_createXAQueueConnection_1(FHandle xaqcf, FHandle xaqcon-
nection, const char username, const char password);
```

Parameters

`xaqcf` - The XA queue connection factory to operate on.

`xaqconnection` - A newly created XA queue connection.

`username` - The caller's username.

`password` - The caller's password.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcf_createXAConnection

Description

Creates an XAConnection with default user identity.

Function Signature

```
jint xaqcf_createXAConnection(FHandle xacf, FHandle result);
```

Parameters

`xacf` - The XAConnectionFactory to operate on.

`result` - A newly created XAConnection.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcf_createXAConnection_1

Description

Creates an XAConnection with specific user identity.

Function Signature

```
jint xaqcf_createXAConnection_1(FHandle xacf, const char username,
const char password, FHandle result);
```

Parameters

`xacf` - The XAConnectionFactory to operate on.

`username` - The caller's username.

`password` - The caller's password.

`result` - A newly created XAConnection.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcf_createQueueConnection

Description

Creates a queue connection with default user identity.

Function Signature

```
jint xaqcf_createQueueConnection(FHandle qcf, FHandle *qc);
```

Parameters

`qcf` - The queue connection factory to operate on.

`qc` - A newly created queue connection.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcf_createQueueConnection_1

Description

Creates a connection with specified user identity.

Function Signature

```
jint xaqcf_createQueueConnection_1(FHandle qcf, FHandle *qc, const
char *username, const char *passwd);
```

Parameters

`cf` - The connection factory to operate on.

`c` - A newly created connection.

`username` - The caller's username.

`passwd` - The caller's password.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcf_createConnection

Description

Creates a connection with default user identity.

Function Signature

```
jint xaqcf_createConnection(FHandle cf, FHandle *c);
```

Parameters

`cf` -The queue connection factory to operate on.

`c` - A newly created queue connection.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqcf_createConnection_1

Description

Creates a queue connection with specified user identity.

Function Signature

```
jint xaqcf_createConnection_1(FHandle cf, FHandle *c, const char
*username, const char *passwd);
```

Parameters

`cf` - The queue connection factory to operate on.

`c` - A newly created queue connection.

`username` - The caller's username.

`password` - The caller's password.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

XATopicConnectionFactory

An XATopicConnectionFactory provides the same create options as a TopicConnectionFactory. This interface is for use by JMS providers to support transactional environments.

xatcf_createXATopicConnection

Description

Creates an XA topic connection with default user identity.

Function Signature

```
jint xatcf_createXATopicConnection(FHandle xatcf, FHandleFunction
Signature xatopicconnection);
```

Parameters

`xatcf` - The XA topic connection factory to operate on.

`xatopicconnection` - A newly created XA topic connection.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcf_createXATopicConnection_1

Description

Creates an XA topic connection with specific user identity.

Function Signature

```
jint xatcf_createXATopicConnection_1(FHandle xatcf, FHandle xatop-
icconnection, const char userName, const char password);
```

Parameters

`xatcf` - The XA topic connection factory to operate on.

`xatopicconnection` - A newly created XA topic connection.

`username` - The caller's username.

`password` - The caller's password.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcf_createXAConnection

Description

Creates an XAConnection with default user identity.

Function Signature

```
jint xatcf_createXAConnection(FHandle xacf, FHandle result);
```

Parameters

`xacf` - The XAConnectionFactory to operate on.

`result` - A newly created XAConnection.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcf_createXAConnection_1

Description

Creates an XAConnection with specific user identity.

Function Signature

```
jint xatcf_createXAConnection_1(FHandle xacf, const char username,
const char password, FHandle result);
```

Parameters

`xacf` - The XAConnectionFactory to operate on.

`username` - The caller's username.

`password` - The caller's password.

`result` - A newly created XAConnection.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcf_createQueueConnection

Description

Creates a queue connection with default user identity.

Function Signature

```
jint xatcf_createQueueConnection((FHandle qcf, FHandle *qc);
```

Parameters

`qcf` - The queue connection factory to operate on.

`qc` - A newly created queue connection.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcf_createQueueConnection_1

Description

Creates a connection with specified user identity.

Function Signature

```
jint xatcf_createQueueConnection_1(FHandle cf, FHandle *c, const
char *username, const char *passwd);
```

Parameters

`cf` - The connection factory to operate on.

`c` - A newly created connection.

`username` - The caller's username.

`passwd` - The caller's password.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcf_createConnection

Description

Creates a connection with default user identity.

Function Signature

```
jint xatcf_createConnection(FHandle cf, FHandle *c);
```

Parameters

`cf` - The queue connection factory to operate on.

`c` - A newly created queue connection.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatcf_createConnection_1

Description

Creates a queue connection with specified user identity.

Function Signature

```
jint xatcf_createConnection_1(FHandle cf, FHandle *c, const char
*username, const char *passwd);
```

Parameters

`cf` - The queue connection factory to operate on.

`c` - A newly created queue connection.

`username` - The caller's username.

`password` - The caller's password.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

XAConnection

The XAConnection interface extends the capability of Connection by providing an XASession. This interface is for use by JMS providers to support transactional environments.

xaconnection_createXASession

Description

Creates a XASession object.

Function Signature

```
jint xaconnection_createXASession(FHandle xac, FHandle result);
```

Parameters

`xac` - The XAConnection to operate on.

`result` - A newly created XASession.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

XAQueueConnection

An XAQueueConnection provides the same create options as QueueConnection. This interface is for use by JMS providers to support transactional environments.

xaqueueconnection_createXAQueueSession

Description

Creates an XA queue session.

Function Signature

```
jint xaqueueconnection_createXAQueueSession(FHandle xaqc, FHandle
xaqsession);
```

Parameters

`xaqc` - The XA queue connection to operate on.

`xaqsession` - A newly created XA queue session.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqueueconnection_start

Description

Start (or restart) a connection's delivery of incoming messages.

Function Signature

```
jint xaqueueconnection_start(FHandle connection);
```

Parameters

`connection` - The connection to operate on.

Returns

`MQ_SUCCESS` if successful, MQ_FAILURE if error.

xaqueueconnection_stop

Description

Used to temporarily stop a connection's delivery of incoming messages.

Function Signature

```
jint xaqueueconnection_stop(FHandle connection);
```

Parameters

`connection` - The connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqueueconnection_close

Description

Close this connection.

Function Signature

```
jint xaqueueconnection_close(FHandle connection);
```

Parameters

`connection` - The connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqueueconnection_createXASession

Description

Creates a XASession object.

Function Signature

```
jint xaqueueconnection_createXASession(FHandle xac, FHandle re-
sult);
```

Parameters

`xac` - The XAConnection to operate on.

`result` - A newly created XASession.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqueueconnection_getClientID

Description

Gets the client identifier for this connection.

Function Signature

```
jint xaqueueconnection_getClientID(FHandle connection, const char
**result);
```

Parameters

`connection` - The connection to operate on.

`result` - Contains the client ID of this connection.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqueueconnection_setClientID

Description

Sets the client identifier for this connection.

Function Signature

```
jint xaqueueconnection_setClientID(FHandle connection, const char
*clientID);
```

Parameters

`connection` - The connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqueueconnection_setExceptionListener

Description

Sets an exception listener for this connection.

Function Signature

```
jint xaqueueconnection_setExceptionListener(FHandle connection,
OnExceptionMethod listener);
```

Parameters

`connection` - The connection to operate on.

`listener` - A pointer to the callback function to invoke when an exception occurs.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqueueconnection_setExceptionListener_1

Description

Sets an exception listener for this connection.

Function Signature

```
jint xaqueueconnection_setExceptionListener_1(FHandle connection,
OnExceptionMethodWithParam listener, void *param);
```

Parameters

`connection` - The connection to operate on.

`listener` - A pointer to the callback function to invoke when an exception occurs.

`param` - A parameter to be used in the callback function.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqueueconnection_createSession

Description

Creates a Session object.

Function Signature

```
jint xaqueueconnection_createSession(FHandle connection, jboolean
transacted, jint acknowledgeMode, FHandle *result);
```

Parameters

`connection` - The connection to operate on.

`transacted` - If TRUE the session is transacted.

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

`result` - Contains a newly created Session object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

XATopicConnection

An XATopicConnection provides the same create options as TopicConnection. This interface is for use by JMS providers to support transactional environments.

xatopicconnection_createXATopicSession

Description

Creates an XA topic session.

Function Signature

```
jint xatopicconnection_createXATopicSession(FHandle xatc, FHandle
xatopicsession);
```

Parameters

`xatc` - The XA topic connection to operate on.

`xatopicsession` - A newly created XA topic session.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicconnection_start

Description

Start (or restart) a connection's delivery of incoming messages.

Function Signature

```
jint xatopicconnection_start(FHandle connection);
```

Parameters

`connection` - The connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicconnection_stop

Description

Used to temporarily stop a connection's delivery of incoming messages.

Function Signature

```
jint xatopicconnection_stop(FHandle connection);
```

Parameters

`connection` - The connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicconnection_close

Description

Close this connection.

Function Signature

```
jint xatopicconnection_close(FHandle connection);
```

Parameters

`connection` - The connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicconnection_createXASession

Description

Creates a XASession object.

Function Signature

```
jint xatopicconnection_createXASession(FHandle xac, FHandle re-
sult);
```

Parameters

`xac` - The XAConnection to operate on.

`result` - A newly created XASession.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicconnection_getClientID

Description

Gets the client identifier for this connection.

Function Signature

```
jint xatopicconnection_getClientID(FHandle connection, const char
**result);
```

Parameters

`connection` - The connection to operate on.

`result` - Contains the client ID of this connection.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicconnection_setClientID

Description

Sets the client identifier for this connection.

Function Signature

```
jint xatopicconnection_setClientID(FHandle connection, const char
*clientID);
```

Parameters

`connection` - The connection to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicconnection_setExceptionListener

Description

Sets an exception listener for this connection.

Function Signature

```
jint xatopicconnection_setExceptionListener(FHandle connection,
OnExceptionMethod listener);
```

Parameters

`connection` - The connection to operate on.

`listener` - A pointer to the callback function to invoke when an exception occurs.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicconnection_setExceptionListener_1

Description

Sets an exception listener for this connection.

Function Signature

```
jint xatopicconnection_setExceptionListener_1(FHandle connection,
OnExceptionMethodWithParam listener, void *param);
```

Parameters

`connection` - The connection to operate on.

`listener` - A pointer to the callback function to invoke when an exception occurs.

`param` - A parameter to be used in the callback function.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicconnection_createSession

Description

Creates a Session object.

Function Signature

```
jint xatopicconnection_createSession(FHandle connection, jboolean
transacted, jint acknowledgeMode, FHandle *result);
```

Parameters

`connection` - The connection to operate on.

`transacted` - If TRUE the session is transacted.

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

`result` - Contains a newly created Session object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

XASession

The XASession interface extends the capability of Session by adding access to a JMS provider's support for the Java Transaction API (JTA).

xasession_getSession

Description

Gets the session associated with this XASession.

Function Signature

```
jint xasession_getSession(FHandle xasession, FHandle result);
```

Parameters

`xasession` - The XA session to operate on.

`result` - Contains the Session associated with this XASession.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xasession_getTransacted

Description

Is the session in transacted mode?

Function Signature

```
jint xasession_getTransacted(FHandle xasession, jboolean result);
```

Parameters

`xasession` - The XA session to operate on.

`result` - Contains the result of the operation (TRUE).

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xasession_getXAResource

Description

Return an XA resource to the caller.

Function Signature

```
jint xasession_getXAResource(FHandle xasession, FHandle xaRe-
source);
```

Parameters

`xasession` - The XA session to operate on.

`xaResource` - Contains the xa resource object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xasession_createBytesMessage

Description

Creates a bytes message.

Function Signature

```
jint xasession_createBytesMessage(FHandle session, FHandle *msg);
```

Parameters

`session` - The session to operate on.

`msg` - A newly created bytes message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xasession_createTextMessage

Description

Creates a text message to be used to send a message containing a string.

Function Signature

```
jint xasession_createTextMessage(FHandle session, FHandle *txtmsg);
```

Parameters

`session` - The session to operate on.

`txtmsg` - A newly created text message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xasession_createTextMessage_1

Description

Creates an initialized text message to be used to send a message containing a string.

Function Signature

```
jint xasession_createTextMessage_1(FHandle session, FHandle *txt-
msg, const char *text);
```

Parameters

`session` - The session to operate on.

`txtmsg` - A newly created text message.

`text` - The string used to initialize this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xasession_recover

Description

Stop message delivery in this session, and restart sending messages with the oldest unacknowledged message.

Function Signature

```
jint xasession_recover(FHandle session);
```

Parameters

`session` - The session to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xasession_setMessageListener

Description

Sets the session's distinguished message listener.

Function Signature

```
jint xasession_setMessageListener(FHandle session, OnMessageMethod
messageListener);
```

Parameters

`session` - The session to operate on.

`messageListener` - The callback function to invoke when a message arrives.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xasession_setMessageListener_1

Description

Sets the session's distinguished message listener.

Function Signature

```
jint xasession_setMessageListener_1(FHandle session, OnMes-
sageMethodWithParam messageListener, void *param);
```

Parameters

`session` - The session to operate on.

`messageListener` - The callback function to invoke when a message arrives.

`param` - A parameter to be used in the callback function.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xasession_createBrowser

Description

Creates a QueueBrowser object to peek at the messages on the specified destination, which must be a queue.

Function Signature

```
jint xasession_createBrowser(FHandle session, FHandle destination,
FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destination` - The Destination to access.

`result` - Contains the newly created QueueBrowser object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xasession_createBrowser_1

Description

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

Function Signature

```
jint xasession_createBrowser_1(FHandle session, FHandle destina-
tion, const char *messageSelector, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destination` - The Destination to access.

`messageSelector` - Only messages with properties matching the message selector expression are delivered.

`result` - Contains the newly created QueueBrowser object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xasession_createConsumer

Description

Creates a MessageConsumer object to receive messages from the specified destination.

Function Signature

```
jint xasession_createConsumer(FHandle session, FHandle destination,
FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destination` - The Destination to access.

`result` - Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xasession_createConsumer_1

Description

Creates a message consumer to the specified destination, using a message selector.

Function Signature

```
jint xasession_createConsumer_1(FHandle session, FHandle destina-
tion, const char *messageSelector, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destination` - The Destination to access.

`messageSelector` - Only messages with properties matching the message selector expression are delivered.

`result` - Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xasession_createConsumer_2

Description

Creates a message consumer to the specified destination, using a message selector.

Function Signature

```
jint xasession_createConsumer_2(FHandle session, FHandle destination, const char *messageSelector, jboolean NoLocal, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destination` - The Destination to access.

`messageSelector` - Only messages with properties matching the message selector expression are delivered.

`NoLocal` - If set, inhibits the delivery of messages published by its own connection.

`result` - Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xasession_createConsumer_3

Description

Creates a durable message consumer of the specified destination The subscription name is used to support durable subscription to topics.

Function Signature

```
jint xasession_createConsumer_3(FHandle session, FHandle destina-
tion, const char *messageSelector, jboolean NoLocal, const char
*subscriptionName, FHandle *result);
```

Parameter

`session` - The Session to operate on.

`destination` - The Destination to access.

`messageSelector` - Only messages with properties matching the message selector expression are delivered.

`NoLocal` - If set, inhibits the delivery of messages published by its own connection.

`subscriptionName` - The name of the subscription. If the Destination is a queue, the JMS Provider shall set this string to null.

`result` - Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xasession_createDestinationIdentity

Description

Creates a destination object given a Destination name and destinationType (JMS_QUEUE or JMS_TOPIC).

Function Signature

```
jint xasession_createDestinationIdentity(FHandle session, const
char *destinationName, jint destinationType, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destinationName` - The name of this Destination.

`destinationType` - The type of this Destination.

`result` - Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xasession_createProducer

Description

Creates a MessageProducer for the specified destination.

Function Signature

```
jint xasession_createProducer(FHandle session, FHandle destination,
FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destination` - The Destination to access.

`result` - Contains the newly created MessageProducer object.

Returns

`MQ_SUCCESS` if successful, MQ_FAILURE if error.

xasession_createTemporaryDestination

Description

Creates a TemporaryDestination object.

Function Signature

```
jint xasession_createTemporaryDestination(FHandle session, jint
destinationType, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destinationType` - The type of this TemporaryDestination.

`result` - Contains the newly created TemporaryDestination object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xasession_getMessageConsumer

Description

Gets a MessageConsumer object based on a Destination and subscriptionName pair.

Function Signature

```
jint xasession_getMessageConsumer(FHandle session, FHandle destina-
tion, const char *subscriptionName, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destination` - The Destination to access.

`subscriptionName` - A string used to identify the subscription.

`result` - Contains a MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

XAQueueSession

An XAQueueSession provides a regular QueueSession, which can be used to create QueueReceiver, QueueSender, and QueueBrowser objects.

xaqueuesession_getQueueSession

Description

Gets the queue session associated with this XA queue session.

Function Signature

```
jint xaqueuesession_getQueueSession(FHandle xaqs, FHandle qs);
```

Parameters

`xaqs` - The XA queue session to operate on.

`qs` - The queue session object.

Returns

`MQ_SUCCESS` if successful, MQ_FAILURE if error.

xaqueuesession_close

Description

Since a provider may allocate some resources on behalf of a Session outside the JVM, clients should close them when they are not needed.

Function Signature

```
jint xaqueuesession_close(FHandle session);
```

Parameters

`session` - The session to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqueuesession_getTransacted

Description

Is the session in transacted mode?

Function Signature

```
jint xaqueuesession_getTransacted(FHandle xasession, jboolean re-
sult);
```

Parameters

`xasession` - The XA session to operate on.

`result` - Contains the result of the operation (TRUE).

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqueuesession_getXAResource

Description

Return an XA resource to the caller.

Function Signature

```
jint xaqueuesession_getXAResource(FHandle xasession, FHandle xaRe-
source);
```

Parameters

`xasession` - The XA session to operate on.

`xaResource` - Contains the xa resource object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqueuesession_getSession

Description

Gets the session associated with this XASession.

Function Signature

```
jint xaqueuesession_getSession(FHandle xasession, FHandle result);
```

Parameters

`xasession` - The XA session to operate on.

`result` - Contains the Session associated with this XASession.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqueuesession_createBytesMessage

Description

Creates a bytes message.

Function Signature

```
jint xaqueuesession_createBytesMessage(FHandle session, FHandle
*msg);
```

Parameters

`session` - The session to operate on.

`msg` - A newly created bytes message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqueuesession_createTextMessage

Description

Creates a text message to be used to send a message containing a string.

Function Signature

```
jint xaqueuesession_createTextMessage(FHandle session, FHandle
*txtmsg);
```

Parameters

`session` - The session to operate on.

`txtmsg` - A newly created text message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqueuesession_createTextMessage_1

Description

Creates an initialized text message to be used to send a message containing a string.

Function Signature

```
jint xaqueuesession_createTextMessage_1(FHandle session, FHandle
*txtmsg, const char *text);
```

Parameters

`session` - The session to operate on.

`txtmsg` - A newly created text message.

`text` - The string used to initialize this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqueuesession_recover

Description

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

Function Signature

```
jint xaqueuesession_recover(FHandle session);
```

Parameters

`session` - The session to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqueuesession_setMessageListener

Description

Sets the session's distinguished message listener.

Function Signature

```
jint xaqueuesession_setMessageListener(FHandle session, OnMes-
sageMethod messageListener);
```

Parameters

`session` - The session to operate on.

`messageListener` - The callback function to invoke when a message arrives.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqueuesession_setMessageListener_1

Description

Sets the session's distinguished message listener.

Function Signature

```
jint xaqueuesession_setMessageListener_1(FHandle session, OnMes-
sageMethodWithParam messageListener, void *param);
```

Parameters

`session` - The session to operate on.

`messageListener` - The callback function to invoke when a message arrives.

`param` - A parameter to be used in the callback function.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqueuesession_createBrowser

Description

Creates a QueueBrowser object to peek at the messages on the specified destination, which must be a queue.

Function Signature

```
jint xaqueuesession_createBrowser(FHandle session, FHandle destination, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destination` - The Destination to access.

`result` - Contains the newly created QueueBrowser object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqueuesession_createBrowser_1

Description

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

Function Signature

```
jint xaqueuesession_createBrowser_1(FHandle session, FHandle destination, const char *messageSelector, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destination` - The Destination to access.

`messageSelector` - Only messages with properties matching the message selector expression are delivered.

`result` - Contains the newly created QueueBrowser object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqueuesession_createConsumer

Description

Creates a MessageConsumer object to receive messages from the specified destination.

Function Signature

```
jint xaqueuesession_createConsumer(FHandle session, FHandle destination, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destination` - The Destination to access.

`result` - Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqueuesession_createConsumer_1

Description

Creates a message consumer to the specified destination, using a message selector.

Function Signature

```
jint xaqueuesession_createConsumer_1(FHandle session, FHandle destination, const char *messageSelector, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destination` - The Destination to access.

`messageSelector` - Only messages with properties matching the message selector expression are delivered.

`result` - Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqueuesession_createConsumer_2

Description

Creates a message consumer to the specified destination, using a message selector.

Function Signature

```
jint xaqueuesession_createConsumer_2(FHandle session, FHandle destination, const char *messageSelector, jboolean NoLocal, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destination` - The Destination to access.

`messageSelector` - Only messages with properties matching the message selector expression are delivered.

`NoLocal` - If set, inhibits the delivery of messages published by its own connection.

`result` - Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqueuesession_createConsumer_3

Description

Creates a durable message consumer of the specified destination The subscription name is used to support durable subscription to topics.

Function Signature

```
jint xaqueuesession_createConsumer_3(FHandle session, FHandle destination, const char *messageSelector, jboolean NoLocal, const char *subscriptionName, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destination` - The Destination to access.

`messageSelector` - Only messages with properties matching the message selector expression are delivered.

`NoLocal` - If set, inhibits the delivery of messages published by its own connection.

`subscriptionName` - The name of the subscription. If the Destination is a queue, the JMS Provider shall set this string to null.

`result` - Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqueuesession_createDestinationIdentity

Description

Creates a destination object given a Destination name and destinationType (JMS_QUEUE or JMS_TOPIC).

Function Signature

```
jint xaqueuesession_createDestinationIdentity(FHandle session,
const char *destinationName, jint destinationType, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destinationName` - The name of this Destination.

`destinationType` - The type of this Destination.

`result` - Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqueuesession_createProducer

Description

Creates a MessageProducer for the specified destination.

Function Signature

```
jint xaqueuesession_createProducer(FHandle session, FHandle desti-
nation, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destination` - The Destination to access.

`result` - Contains the newly created MessageProducer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqueuesession_createTemporaryDestination

Description

Creates a TemporaryDestination object.

Function Signature

```
jint xaqueuesession_createTemporaryDestination(FHandle session,
jint destinationType, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destinationType` - The type of this TemporaryDestination.

`result` - Contains the newly created TemporaryDestination object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaqueuesession_getMessageConsumer

Description

Gets a MessageConsumer object based on a Destination and subscriptionName pair.

Function Signature

```
jint xaqueuesession_getMessageConsumer(FHandle session, FHandle
destination, const char *subscriptionName, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destination` - The Destination to access.

`subscriptionName` - A string used to identify the subscription.

`result` - Contains a MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

XATopicSession

An XATopicSession provides a regular TopicSession. which can be used to create TopicSubscriber and TopicPublisher objects.

xatopicsession_getTopicSession

Description

Gets the topic session associated with this XATopicSession.

Function Signature

```
jint xatopicsession_getTopicSession(FHandle xats, FHandleFunction
Signature ts);
```

Parameters

`xats` - The XA topic session to operate on.

`ts` - The topic session object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicsession_close

Description

Since a provider may allocate some resources on behalf of a Session outside the JVM, clients should close them when they are not needed.

Function Signature

```
jint xatopicsession_close(FHandle session);
```

Parameters

`session` - The session to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicsession_getTransacted

Description

Is the session in transacted mode?

Function Signature

```
jint xatopicsession_getTransacted(FHandle xasession, jboolean result);
```

Parameters

`xasession` - The XA session to operate on.

`result` - Contains the result of the operation (TRUE).

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicsession_getXAResource

Description

Return an XA resource to the caller.

Function Signature

```
jint xatopicsession_getXAResource(FHandle xasession, FHandle xaResource);
```

Parameters

`xasession` - The XA session to operate on.

`xaResource` - Contains the xa resource object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicsession_getSession

Description

Gets the session associated with this XASession.

Function Signature

```
jint xatopicsession_getSession(FHandle xasession, FHandle result);
```

Parameters

`xasession` - The XA session to operate on.

`result` - Contains the Session associated with this XASession.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicsession_createBytesMessage

Description

Creates a bytes message.

Function Signature

```
jint xatopicsession_createBytesMessage(FHandle session, FHandle
*msg);
```

Parameters

`session` - The session to operate on.

`msg` - A newly created bytes message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicsession_createTextMessage

Description

Creates a text message to be used to send a message containing a string.

Function Signature

```
jint xatopicsession_createTextMessage(FHandle session, FHandle
*txtmsg);
```

Parameters

`session` - The session to operate on.

`txtmsg` - A newly created text message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicsession_createTextMessage_1

Description

Creates an initialized text message to be used to send a message containing a string.

Function Signature

```
jint xatopicsession_createTextMessage_1(FHandle session, FHandle
*txtmsg, const char *text);
```

Parameters

`session` - The session to operate on.

`txtmsg` - A newly created text message.

`text` - The string used to initialize this message.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicsession_recover

Description

Stop message delivery in this session, and restart sending messages with the oldest unacknowledged message.

Function Signature

```
jint xatopicsession_recover(FHandle session);
```

Parameters

`session` - The session to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicsession_setMessageListener

Description

Sets the session's distinguished message listener.

Function Signature

```
jint xatopicsession_setMessageListener(FHandle session, OnMes-
sageMethod messageListener);
```

Parameters

`session` - The session to operate on.

`messageListener` - The callback function to invoke when a message arrives.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicsession_setMessageListener_1

Description

Sets the session's distinguished message listener.

Function Signature

```
jint xatopicsession_setMessageListener_1(FHandle session, OnMes-
sageMethodWithParam messageListener, void *param);
```

Parameters

`session` - The session to operate on.

`messageListener` - The callback function to invoke when a message arrives.

`param` - A parameter to be used in the callback function.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicsession_createBrowser

Description

Creates a QueueBrowser object to peek at the messages on the specified destination, which must be a queue.

Function Signature

```
jint xatopicsession_createBrowser(FHandle session, FHandle destina-
tion, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destination` - The Destination to access.

`result` - Contains the newly created QueueBrowser object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicsession_createBrowser_1

Description

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

Function Signature

```
jint xatopicsession_createBrowser_1(FHandle session, FHandle destination, const char *messageSelector, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destination` - The Destination to access.

`messageSelector` - Only messages with properties matching the message selector expression are delivered.

`result` - Contains the newly created QueueBrowser object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicsession_createConsumer

Description

Creates a MessageConsumer object to receive messages from the specified destination.

Function Signature

```
jint xatopicsession_createConsumer(FHandle session, FHandle destination, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destination` - The Destination to access.

`result` - Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicsession_createConsumer_1

Description

Creates a message consumer to the specified destination, using a message selector.

Function Signature

```
jint xatopicsession_createConsumer_1(FHandle session, FHandle destination, const char *messageSelector, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destination` - The Destination to access.

`messageSelector` - Only messages with properties matching the message selector expression are delivered.

`result` - Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicsession_createConsumer_2

Description

Creates a message consumer to the specified destination, using a message selector.

Function Signature

```
jint xatopicsession_createConsumer_2(FHandle session, FHandle destination, const char *messageSelector, jboolean NoLocal, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destination` - The Destination to access.

`messageSelector` - Only messages with properties matching the message selector expression are delivered.

`NoLocal` - If set, inhibits the delivery of messages published by its own connection.

`result` - Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicsession_createConsumer_3

Description

Creates a durable message consumer of the specified destination The subscription name is used to support durable subscription to topics.

Function Signature

```
jint xatopicsession_createConsumer_3(FHandle session, FHandle destination, const char *messageSelector, jboolean NoLocal, const char *subscriptionName, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destination` - The Destination to access.

`messageSelector` - Only messages with properties matching the message selector expression are delivered.

`NoLocal` - If set, inhibits the delivery of messages published by its own connection.

`subscriptionName` - The name of the subscription. If the Destination is a queue, the JMS Provider shall set this string to null.

`result` - Contains the newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicsession_createDestinationIdentity

Description

Creates a destination object given a Destination name and destinationType (JMS_QUEUE or JMS_TOPIC).

Function Signature

```
jint xatopicsession_createDestinationIdentity(FHandle session,
const char *destinationName, jint destinationType, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destinationName` - The name of this Destination.

`destinationType` - The type of this Destination.

`result` - Contains - The newly created MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicsession_createProducer

Description

Creates a MessageProducer for the specified destination

Function Signature

```
jint xatopicsession_createProducer(FHandle session, FHandle desti-
nation, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destination` - The Destination to access.

`result` - Contains the newly created MessageProducer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicsession_createTemporaryDestination

Description

Creates a TemporaryDestination object.

Function Signature

```
jint xatopicsession_createTemporaryDestination(FHandle session,
jint destinationType, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destinationType` - The type of this TemporaryDestination.

`result` - Contains the newly created TemporaryDestination object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xatopicsession_getMessageConsumer

Description

Gets a MessageConsumer object based on a Destination and subscriptionName pair.

Function Signature

```
jint xatopicsession_getMessageConsumer(FHandle session, FHandle
destination, const char *subscriptionName, FHandle *result);
```

Parameters

`session` - The Session to operate on.

`destination` - The Destination to access.

`subscriptionName` - A string used to identify the subscription.

`result` - Contains a MessageConsumer object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

4

Miscellaneous APIs

This chapter describes all the other methods related to security, and accessing resource locally or through the World Wide Web.

Utilities

Throwable

The Throwable class is the superclass of all errors and exceptions in the Java language. Only objects that are instances of this class (or one of its subclasses) are thrown by the Java Virtual Machine or can be thrown by the Java throw statement.

A throwable contains a snapshot of the execution stack of its thread at the time it was created. It can also contain a message string that gives more information about the error. Finally, it can contain a cause that is another throwable that caused this throwable to get thrown.

throwable_getCause

Description

Returns the cause of this throwable or null if the cause is nonexistent or unknown.

Function Signature

```
jint throwable_getCause(FHandle t, FHandle *result);
```

Parameters

`t` - The Throwable object to operate on.

`result` - Contains a Throwable object, that is, the cause of this throwable.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

throwable_getLocalizedMessage

Description

Creates a localized description of this throwable.

Function Signature

```
jint throwable_getLocalizedMessage(FHandle t, const char **result);
```

Parameters

`t -` The Throwable object to operate on.

`result` - Contains the localized description of this throwable.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

throwable_getMessage

Description

Returns the detailed message string of this throwable.

Function Signature

```
jint throwable_getMessage(FHandle t, const char **result);
```

Parameters

`t -` The Throwable object to operate on.

`result -` Contains the detail message string of this Throwable instance.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

throwable_printStackTrace

Description

Prints this throwable and its backtrace to the standard error stream.

Function Signature

```
jint throwable_printStackTrace(FHandle t);
```

Parameters

`t -` The Throwable object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

throwable_toString

Description

Returns a short description of this throwable.

Function Signature

```
jint throwable_toString(FHandle t, const char **result);
```

Parameters

`t -` The Throwable object to operate on.

`result-` Contains a string representation of this Throwable.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

URL

URL represents a Uniform Resource Locator, a pointer to a resource on the World Wide Web. A resource can be something as simple as a file or a directory, or it can be a reference to a more complicated object, such as a query to a database or to a search engine.

createURL

Description

Creates a URL object from the String representation.

Function Signature

```
jint createURL(const char *spec, FHandle *url);
```

Parameters

`spec` - The string to parse as a URL.

`url -` Contains a newly created URL object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

createURL_1

Description

Creates a URL object from the specified protocol, host, port number, and file.

Function Signature

```
jint createURL_1(const char *protocol, const char *host, jint port,
const char *file, FHandle *url);
```

Parameters

`protocol -` The name of the protocol to use.

`host-` The name of the host.

`port -`The port no on the host.

`file -` The file on the host.

`url -` Contains a newly created URL object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

createURL_2

Description

Creates a URL from the specified protocol name, host name, and file name.

Function Signature

```
jint createURL_2(const char *protocol, const char *host, const char
*file, FHandle *url);
```

Parameters

`protocol-` The name of the protocol to use.

`host -` The name of the host.

`file -` The file on the host.

`url -` Contains a newly created URL object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

createURL_3

Description

Creates a URL by parsing the given spec within a specified context.

Function Signature

```
jint createURL_3(FHandle context, const char *spec, FHandle *url);
```

Parameters

`context -` The context in which to parse the specification.

`spec -` The string to parse as a URL.

`url -` Contains a newly created URL object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

url_equals

Description

Compares this URL for equality with another object.

Function Signature

```
jint url_equals(FHandle url, FHandle obj, jboolean *result);
```

Parameters

`url -` The URL object to operate on.

`obj -` The object to compare against.

`result -` Contains TRUE if the objects are the same, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

url_getAuthority

Description

Gets the authority part of this URL.

Function Signature

```
jint url_getAuthority(FHandle url, const char **result);
```

Parameters

`url -` The URL object to operate on.

`result -` Contains the authority part of this URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

url_getDefaultPort

Description

Gets the default port number of the protocol associated with this URL.

Function Signature

```
jint url_getDefaultPort(FHandle url, jint *result);
```

Parameters

`url` - The URL object to operate on.

`result` - Contains the port number.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

url_getFile

Description

Gets the file name of this URL.

Function Signature

```
jint url_getFile(FHandle url, const char *result);
```

Parameters

`url` - The URL object to operate on.

`result` - Contains the file name of this URL, or an empty string if one does not exist.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

url_getHost

Description

Gets the host name of this URL, if applicable.

Function Signature

```
jint url_getHost(FHandle url, const char *result);
```

Parameters

`url` - The URL object to operate on.

`result` - Contains the host name.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

url_getPath

Description

Gets the path part of this URL.

Function Signature

```
jint url_getPath(FHandle url, const char *result);
```

Parameters

`url` - The URL object to operate on.

`result -` Contains the path part of this URL, or an empty string if one does not exist.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

url_getPort

Description

Gets the port number of this URL.

Function Signature

```
jint url_getPort(FHandle url, jint
result);
```

Parameters

`url -` The URL object to operate on.

`result` - Contains the port number, or -1 if the port is not set.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

url_getProtocol

Description

Gets the protocol name of this URL.

Function Signature

```
jint url_getProtocol(FHandle url, const char
pr*result);
```

Parameters

`url` - The URL object to operate on.
`result` - Contains the protocol of this URL.

Returns

`MQ_SUCCESS` if successful, MQ_FAILURE if error.

url_getQuery

Description

Gets the query part of this URL.

Function Signature

```
jint url_getQuery(FHandle url, const char *result);
```

Parameters

`url -` The URL object to operate on.
`result -` Contains the query part of this URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

url_getRef

Description

Gets the anchor (also known as the "reference") of this URL.

Function Signature

```
jint url_getRef(FHandle url, const char *result);
```

Parameters

`url` - The URL object to operate on.

`result` - Contains the anchor of this URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

url_getUserInfo

Description

Gets the userInfo part of this URL.

Function Signature

```
jint url_getUserInfo(FHandle url, const char *result);
```

Parameters

`url` - The URL object to operate on.

`result` - Contains the user info part of this URL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

url_sameFile

Description

Compares two URLs, excluding the fragment component.

Function Signature

```
jint url_sameFile(FHandle url, FHandle other, jboolean result);
```

Parameters

`url` - The URL object to operate on.

`other` - The URL object to compare against.

`result` - Contains TRUE if they reference the same remote object; FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

url_toExternalForm

Description

Constructs a string representation of this URL.

Function Signature

```
jint url_toExternalForm(FHandle url, const char *result);
```

Parameters

`url -` The URL object to operate on.

`result -` A string representation of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

InetAddress

This class represents an Internet Protocol (IP) address. An IP address is either a 32-bit or 128-bit unsigned number used by IP, a lower-level protocol on which protocols like UDP and TCP are built. An instance of an InetAddress consists of an IP address and the corresponding host name.

inetaddress_equals

Description

Compares this object against the specified object.

Function Signature

```
jint inetaddress_equals(FHandle ia, FHandle obj, jboolean result);
```

Parameters

`ia -` The InetAddress object to operate on.

`obj -` The object to compare against.

`result -` Contains TRUE if the objects are the same, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

inetaddress_getAddress

Description

Returns the raw IP address of this InetAddress object.

Function Signature

```
jint inetaddress_getAddress(FHandle ia, jbyte result[]);
```

Parameters

`ia` - The InetAddress object to operate on.

`result` - Contains the raw IP address of this object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

inetaddress_getCanonicalHostName

Description

Gets the fully qualified domain name for this IP address.

Function Signature

```
jint inetaddress_getCanonicalHostName(FHandle ia, const char
*result);
```

Parameters

`ia` - The InetAddress object to operate on.

`result` - Contains the fully qualified domain name for this IP address, or if the operation is not allowed by the security check, the textual representation of the IP address.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

inetaddress_getHostAddress

Description

Returns the IP address string in textual presentation.

Function Signature

```
jint inetaddress_getHostAddress(FHandle ia, const char *result);
```

Parameters

`ia` - The InetAddress object to operate on.

`result` - Contains the raw IP address in string format.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

inetaddress_getHostName

Description

Gets the host name for this IP address.

Function Signature

```
jint inetaddress_getHostName(FHandle ia, const char *result);
```

Parameters

`ia` - The InetAddress object to operate on.

`result` - Contains the host name for this IP address, or if the operation is not allowed by the security check, the textual representation of the IP address.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

inetaddress_isAnyLocalAddress

Description

Utility routine to check if the InetAddress in a wildcard address.

Function Signature

```
jint inetaddress_isAnyLocalAddress(FHandle ia, jboolean result);
```

Parameters

`ia` - The InetAddress object to operate on.

`result` - Contains a boolean indicating if the InetAddress is a wildcard address.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

inetaddress_isLinkLocalAddress

Description

Utility routine to check if the InetAddress is an link local address.

Function Signature

```
jint inetaddress_isLinkLocalAddress(FHandle ia, jboolean result);
```

Parameters

`ia -` The InetAddress object to operate on.

`result -` Contains a boolean indicating if the InetAddress is a link local address; or false if address is not a link local unicast address.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

inetaddress_isLoopbackAddress

Description

Utility routine to check if the InetAddress is a loopback address.

Function Signature

```
jint inetaddress_isLoopbackAddress(FHandle ia, jboolean result);
```

Parameters

`ia -` The InetAddress object to operate on.

`result -` Contains a boolean indicating if the InetAddress is a loopback address.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

inetaddress_isMCGlobal

Description

Utility routine to check if the multicast address has global scope.

Function Signature

```
jint inetaddress_isMCGlobal(FHandle ia, jboolean result);
```

Parameters

`ia -` The InetAddress object to operate on.

`result -` Contains a boolean indicating if the address has is a multicast address of global scope, FALSE if it is not of global scope or it is not a multicast address.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

inetaddress_isMCLinkLocal

Description

Utility routine to check if the multicast address has link scope.

Function Signature

```
jint inetaddress_isMCLinkLocal(FHandle ia, jboolean result);
```

Parameters

`ia -` The InetAddress object to operate on.

`result -` A boolean indicating if the address has is a multicast address of link-local scope, FALSE if it is not of link-local scope or it is not a multicast address.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

inetaddress_isMCNodeLocal

Description

Utility routine to check if the multicast address has node scope.

Function Signature

```
jint inetaddress_isMCNodeLocal(FHandle ia, jboolean result);
```

Parameters

`ia -` The InetAddress object to operate on.

`result` - Contains a boolean indicating if the address has is a multicast address of node-local scope, FALSE if it is not of node-local scope or it is not a multicast address.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

inetaddress_isMCOrgLocal

Description

Utility routine to check if the multicast address has organization scope.

Function Signature

```
jint inetaddress_isMCOrgLocal(FHandle ia, jboolean result);
```

Parameters

`ia` - The InetAddress object to operate on.

`result` - A boolean indicating if the address has is a multicast address of organization-local scope, FALSE if it is not of organization-local scope or it is not a multicast address.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

inetaddress_isMCSiteLocal

Description

Utility routine to check if the multicast address has site scope.

Function Signature

```
jint inetaddress_isMCSiteLocal(FHandle ia, jboolean
result);
```

Parameters

`ia` - The InetAddress object to operate on.

`result` - Contains a boolean indicating if the address has is a multicast address of site-local scope, FALSE if it is not of site-local scope or it is not a multicast address.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

inetaddress_isMulticastAddress

Description

Utility routine to check if the InetAddress is an IP multicast address.

Function Signature

```
jint inetaddress_isMulticastAddress(FHandle ia, jboolean result);
```

Parameters

`ia` - The InetAddress object to operate on.

`result` - Contains a boolean indicating if the InetAddress is an IP multicast address.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

inetaddress_isSiteLocalAddress

Description

Utility routine to check if the InetAddress is a site local address.

Function Signature

```
jint inetaddress_isSiteLocalAddress(FHandle ia, jboolean result);
```

Parameters

`ia` - The InetAddress object to operate on.

`result` - Contains a boolean indicating if the InetAddress is a site local address; or FALSE if address is not a site local unicast address.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

Enumeration

It represents an object that implements the Enumeration interface. This interface generates a series of elements, one at a time. Successive calls to the nextElement method return successive elements of the series.

enumeration_hasMoreElements

Description

Creates a Hashtable with default parameters.

Function Signature

```
jint enumeration_hasMoreElements(FHandle enumeration, jboolean
*result);
```

Parameters

`enumeration -` The enumeration object to operate on.result TRUE if and only if this enumeration object contains at least one more element to provide; FALSE otherwise.

Returns

MQ_SUCCESS If successful, MQ_FAILURE if error.

enumeration_nextElement

Description

Creates a Hashtable with default parameters.

Declaraion

```
jint enumeration_nextElement(FHandle enumeration, FHandle *object);
```

Parameters

`object -` The next element of this enumeration.

`enumeration -` The enumeration object to operate on

Returns

MQ_SUCCESS If successful, MQ_FAILURE if error.

Hashtable

This class implements a hashtable, which maps keys to values. Any non-null object can be used as a key or as a value. To successfully store and retrieve objects from a hashtable, the objects used as keys must implement the hashCode method and the equals method.

createHashtable

Description

Creates a Hashtable with default parameters.

Function Signature

```
jint createHashtable(FHandle *ht);
```

Parameters

`ht` - A newly created Hashtable.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ht_put

Description

Put a key-value pair into a Hashtable.

Function Signature

```
jint ht_put(FHandle ht, const char *key, const char *value);
```

Parameters

`ht` - The Hashtable to operate against.

`key` - The key.

`value` - The value.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

ht_get

Description

Gets the value corresponding to a given key.

Function Signature

```
jint ht_get(FHandle ht, const char *key, const char **value);
```

Parameters

`ht` - The Hashtable to operate against.

`key` - The key.

`value` - Contains the value after the call completes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

hashtable_clear

Description

Clears this hashtable so that it contains no keys.

Function Signature

```
jint hashtable_clear(FHandle ht);
```

Parameters

`ht` - The Hashtable to operate against.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

hashtable_contains

Description

Tests if some key maps into the specified value in this hashtable.

Function Signature

```
jint hashtable_contains(FHandle ht, const char value, jboolean re-
sult);
```

Parameters

`ht` - The Hashtable to operate against.

`value` - The value to check for.

`result` - Contains TRUE if and only if some key maps to the value argument in this hashtable as determined by the equals method; FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

hashtable_containsKey

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

Function Signature

```
jint hashtable_containsKey(FHandle ht, const char key, jboolean result);
```

Parameters

`ht` - The Hashtable to operate against.

`key` - Possible key.

`result` - Contains TRUE if and only if the specified object is a key in this hashtable, as determined by the equals method; FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

hashtable_containsValue

Description

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

Function Signature

```
jint hashtable_containsValue(FHandle ht, const char value, jboolean result);
```

Parameters

`ht` - The Hashtable to operate against.

`value` - The value whose presence in this Hashtable is to be tested.

`result` - Contains TRUE if this map maps one or more keys to the specified value.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

hashtable_elements

Description

Returns an enumeration of the values in this hashtable.

Function Signature

```
jint hashtable_elements(FHandle ht, FHandle result);
```

Parameters

`ht` - The Hashtable to operate against.

`result` - Contains an Enumeration object with all the values in this Hashtable.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

hashtable_isEmpty

Description

Tests if this hashtable maps no keys to values.

Function Signature

```
jint hashtable_isEmpty(FHandle ht, jboolean result);
```

Parameters

`ht` - The Hashtable to operate against.

`result` - Contains TRUE if this hashtable maps no keys to values; FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

hashtable_keys

Description

Returns an enumeration of the keys in this hashtable.

Function Signature

```
jint hashtable_keys(FHandle ht, FHandle result);
```

Parameters

`ht` - The Hashtable to operate against.

`result` - An Enumeration object containing all the keys in this Hashtable.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

hashtable_remove

Description

Removes the key (and its corresponding value) from this hashtable.

Function Signature

```
jint hashtable_remove(FHandle ht, const char key);
```

Parameters

`ht -` The Hashtable to operate against.

`key -` The key needs to be removed.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

hashtable_size

Description

Returns the number of keys in this hashtable.

Function Signature

```
jint hashtable_size(FHandle ht, jint result);
```

Parameters

`ht -` The Hashtable to operate against.

`result -` Contains the number of keys in this Hashtable.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

Security

This class provides the classes and interfaces for the security framework. The varous interfaces in this package include

- Owner
- Acl
- AclEntry
- Principal
- Group
- Permissions

Owner

This interface is used for managing owners of Access Control Lists (ACLs) or ACL configurations. The ACL interface in the java.security.acl package extends this Owner interface.

owner_addOwner

Description

Adds an owner.

Function Signature

```
jint owner_addOwner(FHandle owner, FHandle caller, FHandle obj,
jboolean result);
```

Parameters

`owner` - The Owner object to operate on.

`caller` - The principal invoking this method. It must be an owner of the ACL.

`obj` - The Owner object that should be added to the list of owners.

`result` - Contains TRUE if successful, FALSE if owner is already an owner.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

owner_deleteOwner

Description

Deletes an owner.

Function Signature

```
jint owner_deleteOwner(FHandle owner, FHandle caller, FHandle obj,
jboolean result);
```

Parameters

`owner` - The Owner object to operate on.

`caller` - The principal invoking this method. It must be an owner of the ACL.

`obj` - The Owner object to be removed from the list of owners.

`result` - Contains TRUE if the owner is removed, FALSE if the owner is not part of the list of owners.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

owner_isOwner

Description

Returns TRUE if the given principal is an owner of the ACL.

Function Signature

```
jint owner_isOwner(FHandle owner, FHandle obj, jboolean result);
```

Parameters

`owner` - The Owner object to operate on.

`obj` - The principal to be checked to determine whether or not it is an owner.

`result` - Contains TRUE if the passed principal is in the list of owners, FALSE if not.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

Acl

This interface represents an Access Control List (ACL). An Access Control List is a data structure used to guard access to resources. This data structure contains multiple ACL entries and each ACL entry, of interface type AclEntry, contains a set of permissions associated with a particular principal.

acl_addEntry

Description

Adds an ACL entry to this ACL.

Function Signature

```
jint acl_addEntry(FHandle acl, FHandle caller, FHandle entry, jbool-
ean result);
```

Parameters

`acl` - The Acl object to operate on.

`caller` - The principal invoking this method. It must be an owner of this ACL.

`entry` - The ACL entry to be added to this ACL.

`result` - Contains TRUE on success, FALSE if an entry of the same type (positive or negative) for the same principal is already present in this ACL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acl_checkPermission

Description

Checks whether or not the specified principal has the specified permission.

Function Signature

```
jint acl_checkPermission(FHandle acl, FHandle principal, FHandle
permission, jboolean result);
```

Parameters

`acl` - The Acl object to operate on.

`principal` - The Principal, assumed to be a valid authenticated Principal.

`permission` - The Permission to be checked for.

`result` - Contains TRUE if the principal has the specified permission, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acl_entries

Description

Returns an enumeration of the entries in this ACL.

Function Signature

```
jint acl_entries(FHandle acl, FHandle result);
```

Parameters

`acl` - The Acl object to operate on.

`result` - Contains an Enumeration object of all the entries in this Acl.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acl_getName

Description

Returns the name of this ACL.

Function Signature

```
jint acl_getName(FHandle acl, const char *result);
```

Parameters

`acl` - The Acl object to operate on.

`result` - Contains the name of this Acl.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acl_getPermissions

Description

Returns an enumeration for the set of allowed permissions for the specified principal (representing an entity such as an individual or a group).

Function Signature

```
jint acl_getPermissions(FHandle acl, FHandle user, FHandle result);
```

Parameters

`acl -` The Acl object to operate on.

`user -` The principal whose permission set is to be returned.

`result -` Contains an Enumeration object with the permission set specifying the permissions the principal is allowed.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acl_removeEntry

Description

Removes an ACL entry from this ACL.

Function Signature

```
jint acl_removeEntry(FHandle acl, FHandle caller, FHandle entry,
jboolean result);
```

Parameters

`acl -` The Acl object to operate on.

`caller -` The Principal invoking this method. It must be an owner of this ACL.

`entry -` The AclEntry to be removed from this Acl.

`result -` Contains TRUE on success, FALSE if the entry is not part of this ACL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acl_setName

Description

Sets the name of this ACL.

Function Signature

```
jint acl_setName(FHandle acl, FHandle caller, const char name);
```

Parameters

`acl` - The Acl object to operate on.

`caller` - The Principal invoking this method. It must be an owner of this ACL.

`name` - The name to be given to this ACL.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acl_addOwner

Description

Adds an owner.

Function Signature

```
jint acl_addOwner(FHandle owner, FHandle caller, FHandle obj, jbool-
ean result);
```

Parameters

`owner` - The Owner object to operate on.

`caller` - The principal invoking this method. It must be an owner of the ACL.

`obj` - The Owner object that should be added to the list of owners.

`result` - Contains TRUE if successful, FALSE if owner is already an owner.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acl_deleteOwner

Description

Deletes an owner.

Function Signature

```
jint acl_deleteOwner(FHandle owner, FHandle caller, FHandle obj,
jboolean result);
```

Parameters

`owner` - The Owner object to operate on.

`caller` - The principal invoking this method. It must be an owner of the ACL.

`obj` - The Owner object to be removed from the list of owners.

`result` - Contains TRUE if the owner is removed, FALSE if the owner is not part of the list of owners.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

acl_isOwner

Description

Returns TRUE if the given principal is an owner of the ACL.

Function Signature

```
jint acl_isOwner(FHandle owner, FHandle obj, jboolean result);
```

Parameters

`owner` - The Owner object to operate on.

`obj` - The principal to be checked to determine whether or not it is an owner.

`result` - Contains TRUE if the passed principal is in the list of owners, FALSE if not.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

AclEntry

This interface is used for representing one entry in an Access Control List (ACL). Each ACL entry object contains a set of permissions associated with a particular principal and each ACL entry is specified as being either positive or negative. If negative, the permissions are to be denied. Each principal can have at most one positive ACL entry and one negative entry.

Multiple positive or negative ACL entries are not allowed for any principal

aclentry_addPermission

Description

Adds the specified permission to this ACL entry.

Function Signature

```
jint aclentry_addPermission(FHandle ae, FHandle permission, jboolean
result);
```

Parameters

`ae` - The AclEntry object to operate on.

`permission` - The permission to be associated with the principal in this entry.

`result` - Contains TRUE if the permission was added, FALSE if the permission was already part of this entry's permission set.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

aclentry_checkPermission

Description

Checks if the specified permission is part of the permission set in this entry.

Function Signature

```
jint aclentry_checkPermission(FHandle ae, FHandle permission, jbool-
ean result);
```

Parameters

`ae` - The AclEntry object to operate on.

`permission` - The permission to be checked for.

`result` - Contains TRUE if the permission is part of the permission set in this entry, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

aclentry_clone

Description

Clones this ACL entry.

Function Signature

```
jint aclentry_clone(FHandle ae, FHandle result);
```

Parameters

`ae` - The AclEntry object to operate on.

`result` - Contains a clone of this AclEntry.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

aclentry_getPrincipal

Description

Returns the principal for which permissions are granted or denied by this ACL entry.

Function Signature

```
jint aclentry_getPrincipal(FHandle ae, FHandle result);
```

Parameters

`ae` - The AclEntry object to operate on.

`result` - Contains the Principal associated with this entry.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

aclentry_isNegative

Description

Returns true if this is a negative ACL entry (one denying the associated principal the set of permissions in the entry), false otherwise.

Function Signature

```
jint aclentry_isNegative(FHandle ae, jboolean result);
```

Parameters

`ae` - The AclEntry object to operate on.

`result` - Contains TRUE if this is a negative ACL entry, FALSE if it's not.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

aclentry_permissions

Description

Returns an enumeration of the permissions in this ACL entry.

Function Signature

```
jint aclentry_permissions(FHandle ae, FHandle result);
```

Parameters

`ae` - The AclEntry object to operate on.

`result` - Contains an Enumeration object of the permissions in this ACL entry.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

aclentry_removePermission

Description

Removes the specified permission from this ACL entry.

Function Signature

```
jint aclentry_removePermission(FHandle ae, FHandle permission,
jboolean result);
```

Parameters

`ae` - The AclEntry object to operate on.

`permission` - The Permission to be removed from this entry.

`result` - Contains TRUE if the permission is removed, FALSE if the permission was not part of this entry's permission set.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

aclentry_setNegativePermissions

Description

Sets this ACL entry to be a negative one.

Function Signature

```
jint aclentry_setNegativePermissions(FHandle ae);
```

Parameters

`ae -` The AclEntry object to operate on.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

aclentry_setPrincipal

Description

Specifies the principal for which permissions are granted or denied by this ACL entry.

Function Signature

```
jint aclentry_setPrincipal(FHandle ae, FHandle user, jboolean result);
```

Parameters

`ae -` The AclEntry object to operate on.

`user -` The Principal to be set for this entry.

`result-` Contains TRUE if the principal is set, FALSE if there was already a principal set for this entry.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

Principal

This interface represents the abstract notion of a principal, which can be used to represent any entity, such as an individual, a corporation, and a login id.

principal_equals

Description

Compares this principal to the specified object.

Function Signature

```
jint principal_equals(FHandle p, FHandle another, jboolean result);
```

Parameters

`p -` The Principal object to operate on.

`another -` The object to compare against.

`result -` Contains TRUE if the principal passed in is the same as that encapsulated by this principal, and FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

principal_getName

Description

Returns the name of this principal.

Function Signature

```
jint principal_getName(FHandle p, const char *result);
```

Parameters

`p -` The Principal object to operate on.

`result -` Contains the name of this Principal.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

Group

This interface is used to represent a group of principals.

group_addMember

Description

Adds the specified member to the group.

Function Signature

```
jint group_addMember(FHandle g, FHandle user, jboolean
result);
```

Parameters

`g` - The Group object to operate on.

`user` - The Principal to add to this group.

`result` - Contains TRUE if the member was successfully added, FALSE if the principal was already a member.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

group_isMember

Description

Returns true if the passed principal is a member of the group.

Function Signature

```
jint group_isMember(FHandle g, FHandle member, jboolean result);
```

Parameters

`g` - The Group object to operate on.

`member` - The Principal whose membership is to be checked.

`result` - Contains TRUE if the principal is a member of this group, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

group_members

Description

Returns an enumeration of the members in the group.

Function Signature

```
jint group_members(FHandle g, FHandle result);
```

Parameters

`g -` The Group object to operate on.

`result -` Contains an Enumeration object containing the group members.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

group_removeMember

Description

Removes the specified member from the group.

Function Signature

```
jint group_removeMember(FHandle g, FHandle user, jboolean result);
```

Parameters

`g -` The Group object to operate on.

`user -` The Principal to remove from this group.

`result -` Contains TRUE if the principal was removed, or FALSE if the principal was not a member.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

Permission

This interface represents a permission that is used to grant a particular type of access to a resource

permission_equals

Description

Returns true if the object passed matches the permission represented in this interface.

Function Signature

```
jint permission_equals(FHandle p, FHandle obj, jboolean result);
```

Parameters

`g -` The Group object to operate on.

`obj -` The object to compare against.

`result -` Contains TRUE if the Permission objects are equal, FALSE otherwise.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

Naming and Lookup

These APIs allow a user to create an initial context and lookup admininstered objects such as Topics, Queues, and ConnectionFactories from a naming directory.

InitialContext

This class is the starting context for performing naming operations. All naming operations are relative to a context. The initial context implements the Context interface and provides the starting point for resolution of names.

createInitialContext

Description

Creates an initial context using the supplied environment.

Function Signature

```
jint createInitialContext(FHandle *ic, FHandle env);
```

Parameters

`ic` - A newly created initial context.

`env` - Environment used to create the initial context.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

initialcontext_lookup

Description

Retrieves the named object.

Function Signature

```
jint initialcontext_lookup(FHandle ic, FHandle *cf, const char
*name);
```

Parameters

`ic` - The initial context to operate on.

`name` - Name of the object to lookup.

Returns

MQ_SUCCESS

If successful, MQ_FAILURE if error.

XA

XAResource

The XAResource interface is a Java mapping of the industry standard XA interface based on the X/Open CAE Specification. The XA interface defines the contract between a Resource Manager and a Transaction Manager in a distributed transaction processing (DTP) environment. A JDBC driver or a JMS provider implements this interface to support the association between a global transaction and a database or message service connection.

The XAResource interface can be supported by any transactional resource that is intended to be used by application programs in an environment where transactions are controlled by an external transaction manager.

`XAResource_TMENDRSCAN`

Ends a recovery scan.

`XAResource_TMFAIL`

Disassociates the caller and marks the transaction branch rollback-only.

`XAResource_TMJOIN`

Caller is joining existing transaction branch.

`XAResource_TMNOFLAGS`

Use TMNOFLAGS to indicate that no flags value is selected.

`XAResource_TMONEPHASE`

Caller is using one-phase optimization.

`XAResource_TMRESUME`

Caller is resuming association with a suspended transaction branch.

`XAResource_TMSTARTRSCAN`

Starts a recovery scan.

`XAResource_TMSUCCESS 4`

Disassociates caller from a transaction branch.

`XAResource_TMSUSPEND`

Caller is suspending (not ending) its association with a transaction branch.

`XAResource_XA_OK 0`

The transaction work has been prepared normally.

`XAResource_XA_RDONLY`

The transaction branch has been read-only and has been committed.

xaresource_commit

Description

Commits the global transaction specified by xid.

Function Signature

```
jint xaresource_commit(FHandle xaresource, FHandle xid, jboolean
onePhase);
```

Parameters

`xaresource` - The XA resource to operate on.

`xid` - A global transaction identifier.

`onePhase` - If TRUE, the resource manager should use a one-phase commit protocol to commit the work done on behalf of xid.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaresource_end

Description

Ends the work performed on behalf of a transaction branch.

Function Signature

```
jint xaresource_end(FHandle xaresource, FHandle xid, jint flags);
```

Parameters

`xaresource` - The XA resource to operate on.

`xid` - A global transaction identifier.

`flags` - One of TMSUCCESS, TMFAIL, or TMSUSPEND.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaresource_forget

Description

Tells the resource manager to forget about a heuristically completed transaction branch.

Function Signature

```
jint xaresource_forget(FHandle xaresource, FHandle xid);
```

Parameters

`xaresource` - The XA resource to operate on.

`xid` - A global transaction identifier.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaresource_getTransactionTimeout

Description

Obtains the current transaction timeout value set for this XAResource instance.

Function Signature

```
jint xaresource_getTransactionTimeout(FHandle xaresource, jint val-
ue);
```

Parameters

`xaresource` - The XA resource to operate on.

`value` -The transaction time out value in seconds.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaresource_isSameRM

Description

This method is called to determine if the resource manager instance represented by xaresource1 is the same as the resource manager instance represented by the parameter xaresource2.

Function Signature

```
jint xaresource_isSameRM(FHandle xaresource1, FHandle xaresource2,
jboolean result);
```

Parameters

`xaresource1` - An XA resource.

`xaresource1` - An XA resource.

`result` - True if it's the same RM instance; otherwise false.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaresource_prepare

Description

Ask the resource manager to prepare for a transaction commit of the transaction specified in xid.

Function Signature

```
jint xaresource_prepare(FHandle xaresource, FHandle xid, jint re-
sult);
```

Parameters

`xaresource` - The XA resource to operate on.

`xid` - A global transaction identifier.

`result` - A value indicating the resource manager's vote on the outcome of the transaction. The possible values are: XA_RDONLY or XA_OK.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaresource_recover

Description

Obtains a list of prepared transaction branches from a resource manager.

Function Signature

```
jint xaresource_recover(FHandle xaresource, jint flag, FHandle *xi-
dArray, jint length);
```

Parameters

`xaresource` - The XA resource to operate on.

`flag` - One of TMSTARTRSCAN, TMENDRSCAN, TMNOFLAGS. TMNOFLAGS must be used when no other flags are set in the parameter.

`xidArray` - Zero or more XIDs of the transaction branches that are currently in a prepared or heuristically completed state.

`length` - The length of xidArray.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaresource_rollback

Description

Informs the resource manager to roll back work done on behalf of a transaction branch.

Function Signature

```
jint xaresource_rollback(FHandle xaresource, FHandle xid);
```

Parameters

`xaresource` - The XA resource to operate on.

`xid` - A global transaction identifier.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaresource_setTransactionTimeout

Description

Sets the current transaction timeout value for this XAResource instance.

Function Signature

```
jint xaresource_setTransactionTimeout(FHandle xaresource, jint sec-
onds, jboolean result);
```

Parameters

`xaresource` - The XA resource to operate on.

`seconds` - The transaction timeout value in seconds.

`result` TRUE if the transaction timeout value is set successfully; otherwise FALSE.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xaresource_start

Description

Starts work on behalf of a transaction branch specified in xid.

Function Signature

```
jint xaresource_start(FHandle xaresource, FHandle xid, jint flags);
```

Parameters

`xaresource` - The XA resource to operate on.

`xid` - A global transaction identifier to be associated with the resource.

`flags` - One of TMNOFLAGS, TMJOIN, or TMRESUME.

Returns

MQ_SUCCESS - If successful, MQ_FAILURE if error.

Xid

MAXBQUALSIZE

Description

Maximum number of bytes returned by getBqual.

Function Signature

```
MAXBQUALSIZE
```

MAXGTRIDSIZE

Description

Maximum number of bytes returned by getGtrid.

Function Signature

```
MAXGTRIDSIZE
```

xid_getBranchQualifier

Description

Obtain the transaction branch identifier part of XID as an array of bytes.

Function Signature

```
jint xid_getBranchQualifier(FHandle xid, jbyte *bQualifier, jint
length);
```

Parameters

`xid` - The Xid to operate on.

`bQualifier` - Global transaction identifier.

`length` - The length of the array of bytes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xid_getFormatId

Description

Obtain the format identifier part of the XID.

Function Signature

```
jint xid_getFormatId(FHandle xid, jint formatID);
```

Parameters

`xid` - The Xid to operate on.

`formatID` - Format identifier. O means the OSI CCR format.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

xid_getGlobalTransactionId

Description

Obtain the global transaction identifier part of XID as an array of bytes.

Function Signature

```
jint xid_getGlobalTransactionId(FHandle xid, jbyte *id, jint
length);
```

Parameters

`xid` - The Xid to operate on.

`id` - Global transaction identifier.

`length` - The length of the array of bytes.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

FioranoXid

This class is an implementation of the Xid interface

createFioranoXid

Description

Creates a new FioranoXid object.

Function Signature

```
jint createFioranoXid(const char gtrid, const char bqual, jint for-
matID, FHandle result);
```

Parameters

`gtrid` - The global transaction ID.

`bqual` - The branch qualifier.

`formatID` - The formatID.

`result` - Contains the newly created FioranoXid object.

Returns

MQ_SUCCESS if successful, MQ_FAILURE if error.

5

Using the Sample Programs

This chapter explains the various steps involved in running the sample programs which are shipped as part of the installer.

Organization of Samples Provided

The samples programs illustrating the use of CRTL for PubSub and PTP and XA Operations.

- **pubsub** This directory contains the following sample programs, which illustrate basic JMS Publish/Subscribe functionality, using the CRTL.
- **ptp** This directory contains two sample programs which illustrate basic JMS Send/Receive functionality using the FioranoMQ C Runtime Library.
- **AdminAPIs** The examples in this directory illustrate the use of the FMQ Administration APIs for creation of JMS Administered objects such as Destinations, ConnectionFactories and Users.
- **Unified Domain** The examples in this directory illustrate the use of the FMQ Administration APIs for creation of UnifiedConnectionFactories.
- **XA** The example in this directory illustrates XA capabilities of FioranoMQ and the use of the FMQ Administration APIs for creation of JMS Administered objects such as XA-enabled-Destinations and XA-ConnectionFactories.

Compiling and Running the Samples

Before running the samples, please check if the following has been set in the system path.

- fmq-cpp-client-msg-adapter.dll

- fmq-crosscomp-crtl.dll

To run the samples using FMQ, compile each of the source files using the script file, cclientbuild.bat, in the scripts folder. For convenience, compiled version of the sources are included in every directory.

For information on compiling and running these samples please refer to the readme file in the c\jni\samples directory of FMQ installation.