Sun ONE Message Queue C-API Software Architecture Document

Version 4.2.0

Sun Confidential – For Internal Use Only

SOFTWARE ARCHITECTURE DOCUMENT

1.Introduction

This document describes the design of a MQ C-API that can interact with a MQ broker. Initially it was designed to be used within Sun ONE Directory Server (DS) to process DSML requests sent over the MQ bus. Much of the text in this document was adapted from text appearing in MQ internal documents or the JMS class specification. We are grateful to the writers of these documents.

1.1Design Considerations

The original design of the C-API was guided by the following

- Keep it simple
- Follow the design of the Java client
- Only implement what is needed by DS but facilitate extending it to the full API
- Leave resource allocation decisions to the library user

1.2Client Simplifications

The initial release of MQ C-API will support the following features of Java Message Service

- publish/subscribe and point-to-point communication
- synchronous receive
- asynchronous receive (via listener callback)
- client, auto and dups_ok acknowledgement modes
- JMS transaction
- session recover
- text messages
- bytes message (simplified)
- temporary topics or queues
- selectors

and will support TCP and SSL transports, not support

- map, stream, or object types
- queue browsers
- JMS application server facilities (ConnectionConsumer, Distributed Transaction)

and HTTP transport .

1.3Version

Starting from MQ 4.2, the version of this document will use the corresponding MQ release version.

2.C Client API

This section describes the types and methods associated with the MQ C Client API.

2.1MQ C-API Types

The C Client API defines the following types. The MQ integer types will be defined using C99 standard stdint.h or inttypes.h or platform sized integer types.

MQ Type	Purpose
MQStatus	All methods return a MQStatus struct. The user should not modify or attempt to free the MQStatus struct. MQStatus can be passed to MQ_statusIsError to determine if the status represents an error, MQ_getStatusCode to convert the status to an error code, and MQ_getStatusString to convert it to an error string.
MQConnectionHandle	A handle used to control/refer to a MQ connection. The library maintains a mapping of handles to pointers to objects. The indirection prevents the caller from directly accessing MQ client data structure
MQSessionHandle	A handle used to control/refer to a MQ session.
MQProducerHandle	A handle used to control/refer to a MQ producer.
MQConsumerHandle	A handle used to control/refer to a MQ consumer.
MQDestinationHandle	A handle used to control/refer to a MQ destination.
MQMessageHandle	A handle used to control/refer a MQ message
MQPropertiesHandle	MQPropertiesHandle is a handle to a map from MQStrings to MQBool, MQInt8, MQInt16, MQInt32, MQInt64, MQFloat32, MQFloat64, and MQString. It is used to pass optional parameters to the MQ client methods, and to set/get the header and property fields of messages. Utilities are provided to create and access the properties.
MQ_INVALID_HANDLE	An invalid HANDLE value, which can be used to initialize any of above handles
MQChar	char type
MQBool	It can only assume values MQ_TRUE(=1) and MQ_FALSE(=0).
MQInt8	An 8-bit signed integer
MQInt16	A 16-bit signed integer
MQInt32	A 32-bit signed integer
MQUint32	A 32-bit unsigned integer
MQInt64	A 64-bit signed integer
MQFloat32	A 32-bit floating point number (note: may not be portable)
MQFloat64	A 64-bit floating point number
MQString	A null terminated UTF-8 encoded character string
ConstMQString	A const MQString
MQError	A 32-bit unsigned integer
MQType	MQType is used to return the type of a single property. It's an enum with the following values: MQ_BOOL_TYPE, MQ_INT8_TYPE, MQ_INT16_TYPE, MQ_INT32_TYPE, MQ_INT64_TYPE, MQ_FLOAT32_TYPE, MQ_FLOAT64_TYPE, MQ_STRING_TYPE, MQ_INVALID_TYPE.
MQMessageType	MQMessageType is an enum that is used to determine the type of a message. Currently this can only be MQ_TEXT_MESSAGE or MQ_BYTES_MESSAGE or MQ_MESSAGE otherwise MQ_UNSUPPORTED_MESSAGE
MQAckMode	MQAckMode is an enum of MQ_AUTO_ACKNOWLEDGE, MQ_CLIENT_ACKNOWLEDGE, MQ_DUPS_OK_ACKNOWLEDGE and MQ_SESSION_TRANSACTED

 ©Sun Microsystems, Inc 2001

MQReceiveMode	MQReceiveMode is an enum of values MQ_SESSION_SYNC_RECEIVE and MQ_SESSION_ASYNC_RECEIVE
MQDeliveryMode	MQ message delivery mode. It is an enum of values MQ_NON_PERSISTENT_DELIVERY MQ_PERSISTENT_DELIVERY
MQDestinationType	MQDestinationType is an enum of values MQ_QUEUE_DESTINATION, MQ_TOPIC_DESTINATION
MQLoggingLevel	(**private**) MQLoggingLevel is used to specify the type of information being logged. It is an enum that takes values: MQ_LOG_OFF <= MQ_LOG_SEVERE <= MQ_LOG_WARNING <= MQ_LOG_INFO <= MQ_LOG_CONFIG <= MQ_LOG_FINE <= MQ_LOG_FINER <= MQ_LOG_FINEST MQ_LOG_SEVERE - Indicates serious failure which prevents application normal execution MQ_LOG_WARNING - Indicates a potential problem which will be of interest to end users MQ_LOG_INFO - Information messages that reasonable significant to end users MQ_LOG_CONFIG - Static configuration messages (not used) MQ_FINE - Tracing information that will be broadly interesting to developers, e.g. minor (recoverable) failures, performance related MQ_FINER - Fairly detailed tracing messages, eg. entering, leaving, error as occurred in lower layer functions in the library MQ_FINEST - Highly detailed tracing messages

MQConnectionException-ListenerFunc	MQConnectionExceptionListenerFunc is the type of the callback function that is used to inform the library user that a connection exception has occurred. The connection exception listener is installed at the time MQCreateConnection . The connection exception listener function will only be called synchronously with respect to a single connection. However, if it is installed as the exception listener for multiple connections, it must be reentrant. It has the following definition.
	
	`void (*MQConnectionExceptionListenerFunc)(`
	`const MQConnectionHandle connectionHandle,`
	`MQStatus exception,`
	`void * callbackData )`
	
	connectionHandle is a handle to the connection on which the exception occurred
	
	exception is the connection exception that occurred. Calls to the MQStatus functions can be used to extract more information.
	
	callbackData is whatever void * exceptionCallbackData pointer that was passed to MQCreateConnection

MQMessageListenerFunc	MQMessageListenerFunc is the type of the callback function for asynchronous message receiving. It has the following definition
	
	`MQError (*MQMessageListenerFunc)(`
	`  const MQSessionHandle  sessionHandle,`
	`  const MQConsumerHandle  consumerHandle,`
	`  MQMessageHandle  messageHandle,`
	`  void *  callbackData)`
	
	consumerHandle is the handle to the consumer that the message delivers to and sessionHandle is the handle to the session that the consumer created in.
	
	messageHandle is the handle to the message to be delivered. It's the C-API client's responsibility to free the messageHandle by calling MQFreeMessage.
	
	The callbackData is whatever void * pointer that is passed to the asynchronous consumer creation function.
	
	It's possible for a MQMessageListenerFunc to return a non MQ_SUCCESS error code, however this is considered a client programming error.A well behaved MQMessageListenerFunc should always return MQ_SUCESS and attempt divert the offending message to some application-specific unprocessible message' destination.

MQMessageListenerBAFunc	MQMessageListenerBAFunc is the type of the callback functions of before/after MQMessageListenerFunc for asynchronous message receiving from a XA session. It has the following definition
	
	`MQError (*MQMessageListenerBAFunc)(`
	`  const MQSessionHandle  sessionHandle,`
	`  const MQConsumerHandle  consumerHandle,`
	`  const MQMessageHandle  messageHandle,`
	`  MQError  errorCode`
	`  void *  callbackData)`
	
	consumerHandle is the handle to the consumer that the message will be delivered to and sessionHandle is the handle to the session that the consumer created in. The sessionHandle, consumerHandle and messageHandle are for read-only purpose. errorCode is the client runtime processing status that is passed to the before/after callback functions.

MQMessageArrivedFunc	(**private**) MQMessageArrivedFunc is the type of the callback function that a **synchronous** MessageConsumer uses to receive notification that a message has arrived. This method should return quickly and not attempt to receive/process the message that just arrived. It has the following definition.
	
	`void (*MQMessageArrivedFunc)(`
	`  const MQSessionHandle  sessionHandle,`
	`  const MQConsumerHandle  consumerHandle,`
	`  void *  callbackData )`
	
	sessionHandle is a handle to the session containing the consumer that a message has arrived for.
	
	consumerHandle is a handle to the consumer for which a message was arrived.
	
	callbackData is whatever void * pointer that was passed to the MQSetMessageArrivedFunc.

MQThreadFunc	(**private**) MQThreadFunc is the type of the thread starting function for MQCreateThreadFunc, void (* MQThreadFunc) (void *arg)

MQCreateThreadFunc	(**private**) MQCreateThreadFunc is the type of the callback function that can be passed to MQ_setCreateThreadFunc that allows the client to control how threads are created. It has the following definition:
	
	`MQBool (*MQCreateThreadFunc)(`
	`  MQThreadFunc  startFunc,`
	`  void*  arg,`
	`  void*  callbackData )`
	
	startFunc is the starting function of the thread
	
	arg is the argument to supply to startFunc
	
	callbackData is whatever void* pointer that was passed to MQ_setCreateThreadFunc.
	
	this function should return MQ_TRUE if the thread was created successfully, and MQ_FALSE if the thread could not be created.

MQLoggingFunc	(**private**)MQLoggingFunc is the type of the logging callback function that can be installed so the user has control over what information, generated by the MQ library, is logged where. The MQ client library will ensure that the logging callback is only called synchronously. It has the following definition.
	
	`void (*MQLoggingFunc)(`
	`  MQLoggingLevel severity,`
	`  MQInt32 logCode,`
	`  ConstMQString logMessage,`
	`  MQInt64 timeOfMessage,`
	`  MQInt32 connectionID,`
	`  ConstMQString filename,`
	`  MQInt32 fileLineNumber`
	`  void* userData)`
	
	severity is the severity of the log message.
	
	logCode is the error code associated with this log message.
	
	logMessage is a log message to be logged. logMessage should not be accessed after the logging function returns.
	
	timeOfMessage is the time that the log message was generated. It is represented as the number of microseconds since midnight (00:00:00) 1 January 1970 Coordinated Universal Time (UTC). Note: there is no notion of time zone.
	
	connectionID is the ID of the connection that generated the logging message. If the log message cannot be associated with a connection this field will have value 0.
	
	filename is the name of the source file that generated the log message. This is primarily useful for debugging purposes.
	
	fileLineNumber is the line number of the source file that generated the log message. This is primarily useful for debugging purposes.
	
	callbackData is whatever void* pointer that was supplied to the MQ_setLoggingFunc function.

2.2MQ C-API Methods

MQ C client methods that follows are roughly grouped according to purpose. Except where noted, each method returns a `MQStatus` struct, which can be used to determine if the call was successful and get more information if the call failed. Some of these methods may require additional property fields. The keyword const is used whenever the method will not modify the parameter or in the case of handles, the object the parameter controls.

2.2.1Connection

```
MQStatus MQCreateConnection(
            MQPropertiesHandle                propertiesHandle,
            ConstMQString                     usename,
            ConstMQString                     password,
            ConstMQString                     clientID,
           MQConnectionExceptionListenerFunc exceptionListener,
```

```
            void *                              exceptionCallBackData,
            MQConnectionHandle *                connectionHandle )
```

valid property keys:

KEY	TYPE	VALUES	DEFAULT
MQ_CONNECTION_TYPE_PROPERTY	MQString	"TCP" or "SSL"	"TCP"
MQ_ACK_TIMEOUT_PROPERTY	MQInt32	>=0 in millisec	0(no timeout)
MQ_BROKER_HOST_PROPERTY	MQString	(required)	
MQ_BROKER_PORT_PROPERTY	MQInt32	(required)	
MQ_ACK_ON_PRODUCE_PROPERTY	MQBool		persistent:MQ_TRUE non-persistent:MQ_FALSE
MQ_ACK_ON_ACKNOWLEDGE_PROPERTY	MQBool		MQ_TRUE
MQ_CONNECTION_FLOW_COUNT_PROPERTY	MQInt32		100
MQ_CONNECTION_FLOW_LIMIT_ENABLED_PROPERTY	MQBool		MQ_FALSE
MQ_CONNECTION_FLOW_LIMIT_PROPERTY	MQInt32		1000
MQ_SSL_BROKER_IS_TRUSTED	MQBool		MQ_TRUE
MQ_SSL_CHECK_BROKER_FINGERPRINT	MQBool		MQ_FALSE
MQ_SSL_BROKER_CERT_FINGERPRINT	MQString		NULL

```
NOTE:
. In the first release, the C-API will support MQ 3.5 producer-based flowcontrol
(no new properties need to be exposed in client) and connection-based flowcontrol
for consumers. MQ 3.5 consumer-based flowcontrol will not be supported in this
release. However the C-API will support consumer-based flowcontrol at protocol
level: C-API will always set JMQSize to -1 in consumer RESUME_FLOW (JMQConsumerID
set) and always sending consumer RESUME_FLOW when see C bit in incoming Message
```

This creates a connection to the broker. The clientID parameter can be NULL. The exceptionListener parameter allows a client to be asynchronously notified of a connection problem. Some Connections only consume messages so they would have no other way to learn their Connection has failed. exceptionListener is the connection exception listener callback that is called when a connection exception occurs. exceptionCallbackData is a void * pointer that to be passed to the connection exception listener.

```
MQStatus MQGetXAConnection(
            MQConnectionHandle *                connectionHandle )
```

This gets a XA connection. This should only be called when MQ C-API is used in a X/Open distributed transaction processing environment with MQ as a XA-compliant resource manager. MQCloseConnection should not be called on a XA connection handle.

```
MQStatus MQCreateConnectionExt(
            MQPropertiesHandle                  propertiesHandle,
            ConstMQString                       usename,
             ConstMQString                       password,
             ConstMQString                       clientID,
            MQConnectionExceptionListenerFunc   exceptionListener,
            void *                              exceptionCallbackData,
            MQCreateThreadFunc                  createThreadFunc,
            void *                              createThreadFuncData
            MQBool                              isXA
            MQConnectionHandle *                connectionHandle )
```

(private) same as MQCreateConnection with additional parameters to allow caller specify thread creation function and whether to create a XA or non-XA connection. It returns error if createThreadFunc is NULL.

```
MQStatus MQStartConnection( const MQConnectionHandle connectionHandle )
```

This starts (or restarts) a Connection's delivery of incoming messages.

```
MQStatus MQStopConnection( const MQConnectionHandle connectionHandle )
```

This is used to temporarily stop a Connection's delivery of incoming messages. It can be restarted using MQStartConnection. When stopped, delivery to all the Connection's message consumers is inhibited. Stopping a Connection has no affect on its ability to send messages. Stopping a stopped connection is ignored.

The MQStopConnection method does not return until delivery of messages has paused. The receive timers for a stopped connection continue to advance so receives may time out while the connection is stopped.

```
MQStatus MQCloseConnection( const MQConnectionHandle  connectionHandle )
```

There is no need to close the sessions, producers, and consumers of a closed connection. When this message is invoked it will not return until message processing has been orderly shut down. This will force all threads associated with this connection that are blocking in the library (e.g. a consumer calling MQReceiveMessageWait) to return. This does not actually release all resources associated with the connection. After all of the application threads associated with this connection and its decendant sessions, producers, consumers, etc. have returned, then the application can call MQFreeConnection to release all resources held by this connection and its decendant sessions, producers, consumers, etc..

Closing a connection does NOT force an acknowledge of client acknowledged sessions. Invoking MQ_acknowledgeMessage on a closed connection's session will return an error. Closing a closed connection has no effect.

```
MQStatus MQGetConnectionProperties( const MQConnectionHandle  connectionHandle,
                                    MQPropertiesHandle * propertiesHandle )
```

This gets the connection properties that used to create the connection specified by the connectionHandle. The caller is responsible to free the returned connection properties by MQFreeProperties.

```
MQStatus MQInitializeSSL( ConstMQString  certificateDatabasePath )
```

This function initializes the SSL library. It must be called before connecting to the broker over SSL.

```
MQStatus MQGetMetaData( const MQConnectionHandle   connectionHandle,
                        MQPropertiesHandle *       propertiesHandle)
```

This gets the metadata for this connection. The following metadata property names are supported

KEY	TYPE	VALUE (initial release)
MQ_NAME_PROPERTY	MQString	"Sun ONE Message Queue, Sun Microsystems, Inc."
MQ_VERSION_PROPERTY	MQString	“3.5”
MQ_MAJOR_VERSION_PROPERTY	MQInt32	3
MQ_MINOR_VERSION_PROPERTY	MQInt32	5
MQ_MICRO_VERSION_PROPERTY	MQInt32	0

2.2.2Session

```
MQStatus MQCreateSession(  const MQConnectionHandle   connectionHandle,
                           MQBool                     isTransacted,
                           MQAckMode                  acknowledgeMode,
                           MQReceiveMode              receiveMode,
                           MQSessionHandle *          sessionHandle )
```

This creates a newly created session. A session is a single-thread context for producing and consuming messages. Although a session may create multiple producers and consumers , they are restricted to serial use. In effect, only a single logical thread of control can use them. The acknowledgeMode can be either MQ_AUTO_ACKNOWLEDGE or MQ_CLIENT_ACKNOWLEDGE or MQ_DUPS_OK_ACKNOWLEDGE. If isTransacted parameter is MQ_TRUE, the acknowledgeMode will be ignored. If the session is created with receiveMode MQ_SESSION_SYNC_RECEIVE, only synchronous message consumers (use MQCreateMessageConsumer or MQCreateDurableMessageConsumer) can be created on the session; if the session is created with receiveMode MQ_SESSION_ASYNC_RECEIVE, only asynchronous message consumers (use MQCreateAsyncMessageConsumer or MQCreateAsyncDurableMessageConsumer) can be created on the session. The number of sessions created within a connection is limited only by available system resources.

```
MQStatus MQCreateXASession(
            const MQConnectionHandle    connectionHandle,
            MQReceiveMode               receiveMode,
            MQMessageListenerBAFunc     beforeMessageListener,
            MQMessageListenerBAFunc     afterMessageListener,
            void *                      callbackData,
            MQSessionHandle *           sessionHandle )
```

This creates a newly created XA session. The connectionHandle must be a XA connection handle. A XA session is same as a regular session created by MQCreateSession except:

- A XA session is always XA transacted and the XA transaction is managed by a X/Open distributed transaction manager. MQCommitSession and MQRollbackSession should not be called on a XA session.
- Sending/receiving messages with a XA session must be done in a XA transaction
- If receiveMode is MQ_SESSION_ASYNC_RECEIVE, callback functions beforeMessageListener and afterMessageListener must be specified. beforeMessageListener will be called by the C-API runtime before it calls the messageListener callback; afterMessageListener will be called by the C-API runtime after it calls the messageListener callback. The beforeMesageListener and afterMessageListener functions are provided to the application to associate and disassociate the C-API runtime calling thread with a XA transaction and to demarcate XA transactions, and to set appropriate application association context to the calling thread if the application's distributed transaction processing environment requires that. The beforeMessageListener callback can return an error to the C-API runtime, however like the MQMessageListenerFunc callback, a well programmed application, should not return runtime error from afterMessageListener callback. On calling afterMessageListener callback, the C-API runtime passes an errorCode to the callback to indicate whether the processing of the message acknowledgement successful (see MQMessageListenerBAFunc type). The callbackData parameter will be passed to the beforeMessageListener and afterMessageListener callback functions unchanged.

```
MQStatus MQCloseSession( MQSessionHandle  sessionHandle )
```

There is no need to close the producers/consumers of a closed session. The sessionHandle will be invalid after this call.

```
MQStatus MQCreateDestination( const MQSessionHandle  sessionHandle,
```

```
                                ConstMQString         destinationName,
                                MQDestinationType      destinationType,
                                MQDestinationHandle *  destinationHandle )
```

This creates a destination with a given name and type. Note that this method is not for creating the physical destination. The physical creation of destination is an administration task. The caller is responsible to free the destionationHandle.

```
MQStatus MQCreateTemporaryDestination(
            const MQSessionHandle  sessionHandle,
            MQDestinationType      destinationType,
            MQDestinationHandle *  destinationHandle)
```

This creates a temporary destination of the given type. A temporary destination is a system generated destination that has the duration of the connection that the session belongs to (represented by the sessionHandle). A temporary destination can only be consumed by consumers on the same connection. The caller is responsible to free the destinationHandle.

```
MQStatus MQCreateMessageProducer( const MQSessionHandle     sessionHandle,
                                  MQProducerHandle *        producerHandle )
```

This creates a message producer with no default destination.

```
MQStatus MQCreateMessageProducerForDestination(
                           const MQSessionHandle      sessionHandle,
                           const MQDestinationHandle  destinationHandle,
                           MQProducerHandle *         producerHandle )
```

This creates a message producer with the given destination. The destinationHandle is still valid after the call.

```
MQStatus MQCloseMessageProducer( MQProducerHandle  producerHandle )
```

This closes the message producer.

```
MQStatus MQCreateMessageConsumer(
                           const MQSessionHandle      sessionHandle,
                           const MQDestinationHandle  destinationHandle,
                           ConstMQString              messageSelector,
                           MQBool                     noLocal,
                           MQConsumerHandle *         consumerHandle )
```

This creates a message consumer to the specified destination. The session must created with MQ_SESSION_SYNC_RECEIVE receive-mode. The destinationHandle is still valid after the call.

When the specified destination is a Topic, the message consumer (subscriber) created with MQCreateMessageConsumer only receives messages that are published while it is active. MQCreateDurableMessageConsumer or MQCreateAsyncDurableMessageConsumer can be used to receive messages published while the subscriber is not active. In some cases, a connection may both publish and subscribe to a Topic. The noLocal argument allows a subscriber to inhibit the delivery of messages published by its own connection. Only messages with properties matching the message selector expression are delivered. A value of NULL or an empty string indicates that there is no message selector for the message consumer. A message selector is a MQString whose syntax is based on a subset of the SQL92 conditional expression syntax.

```
MQStatus MQCreateDurableMessageConsumer(
```

```
            const MQSessionHandle      sessionHandle,
            const MQDestinationHandle  destinationHandle,
            ConstMQString              durableName,
            ConstMQString              messageSelector,
            MQBool                     noLocal,
            MQConsumerHandle *         consumerHandle )
```

This creates a durable message consumer with the given durable name to the specified destination. The destination must be a Topic type. The session must created with MQ_SESSION_SYNC_RECEIVE receive-mode. The destinationHandle is still valid after the call.

If a client needs to receive all the messages published on a Topic including the ones published while the subscriber is inactive, it uses this method to create a durable Topic subscriber. MQ retains a record of this durable subscription and insures that all messages from the Topic's publishers are retained until they are either acknowledged by this durable subscriber or they have expired. Sessions with durable subscribers must always provide the same client identifier. In addition, each client must specify a durableName which uniquely identifies (within client identifier) each durable subscription it creates.

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

```
MQStatus MQCreateAsyncMessageConsumer(
            const MQSessionHandle       sessionHandle,
            const MQDestinationHandle   destinationHandle,
            ConstMQString               messageSelector,
            MQBool                      noLocal,
            MQMessageListenerFunc       messageListener,
            void *                      messageListenerCallbackData,
            MQConsumerHandle     *      consumerHandle)
```

This creates a message consumer for asynchronous receiving. The session must be created with MQ_SESSION_ASYNC_RECEIVE receive-mode. The destinationHandle is still valid after the call. The messageListenerCallbackData is a void * pointer that to be passed to the messageListener callback function. The connection must be in stopped mode to call this function because a session with asynchronous receiving mode is dedicated to the thread of control that delivers messages to the message listener callbacks.

```
MQStatus MQCreateAsyncDurableMessageConsumer(
            const MQSessionHandle      sessionHandle,
            const MQDestinationHandle  destinationHandle,
            ConstMQString              durableName,
            ConstMQString              messageSelector,
            MQBool                     noLocal,
            MQMessageListenerFunc      messageListener,
            void *                     messageListenerCallbackData,
            MQConsumerHandle  *        consumerHandle )
```

This creates a durable message consumer for asynchronous receiving. The session must be created with MQ_SESSION_ASYNC_RECEIVE receive-mode. The destination must be a Topic type. The destinationHandle is still valid after the call. The connection must be in stopped mode to call this function because a session with asynchronous receiving mode is dedicated to the thread of control that delivers messages to the message listener callbacks.

```
MQStatus MQCloseMessageConsumer( MQConsumerHandle  consumerHandle )
```

This closes the message consumer.

```
MQStatus MQUnsubscribeDurableMessageConsumer(
              const MQSessionHandle  sessionHandle,
              ConstMQString          durableName )
```

This unsubscribes a durable subscription that has been created by a client. This deletes the state being maintained on behalf of the subscriber by the broker. It is erroneous for a client to delete a durable subscription while it has an active Topic subscriber for it, or while a message received by it has not been acknowledged in the session.

```
MQStatus MQRecoverSession( const MQSessionHandle sessionHandle )
```

This stops message delivery in this session, and restarts message delivery with the oldest unacknowledged message. This method can not be called on a transacted session.

All consumers deliver messages in a serial order. Acknowledging a received message automatically acknowledges all messages that have been delivered to the client.

Restarting a session causes it to take the following actions:
. Stop message delivery
. Mark all messages that might have been delivered but not acknowledged as "redelivered"
. Restart the delivery sequence including all unacknowledged messages that had been previously delivered.
Redelivered messages do not have to be delivered in exactly their original delivery order.

```
MQStatus MQCommitSession( const MQSessionHandle sessionHandle )
```

This commits all messages done in this transaction.

A session may be specified as transacted. Each transacted session supports a single series of transactions. Each transaction groups a set of message sends and a set of message receives into an atomic unit of work. In effect, transactions organize a session's input message stream and output message stream into series of atomic units. When a transaction commits, its atomic unit of input is acknowledged and its associated atomic unit of output is sent. If a transaction rollback is done, the transaction's sent messages are destroyed and the session's input is automatically recovered.

The content of a transaction's input and output units is simply those messages that have been produced and consumed within the session's current transaction.

A transaction is completed using either its session's commit method or its session's rollback method. The completion of a session's current transaction automatically begins the next. The result is that a transacted session always has a current transaction within which its work is done.

```
MQStatus MQRollbackSession( const MQSessionHandle sessionHandle )
```

This rolls back any messages done in this transaction.

```
MQStatus MQGetAcknowledgeMode( const MQSessionHandle sessionHandle,
                               MQAckMode *ackMode  )
```

This gets a session's acknowledge mode. If the session is transacted, MQ_SESSION_TRANSACTED will be returned in the output parameter ackMode.

2.2.3Sending and Receiving

```
MQStatus MQSendMessage( const MQProducerHandle   producerHandle,
                        const MQMessageHandle    messageHandle)
```

This sends the message to the destination using default deliver-mode (MQ_PERSISTENT_DELIVERY), priority (4) and time-to-live (0, forever) for which the specified producer was created. This call can only be used by producers created with MQCreateMessageProducerWithDestination.

```
MQStatus MQSendMessageExt(
                    const MQProducerHandle   producerHandle,
                    const MQMessageHandle    messageHandle,
                    MQDeliveryMode           msgDeliveryMode,
                    MQInt8                   msgPriority,
                    MQInt64                  msgTimeToLive )
```

This is same as MQ_sendMessage except that it allows to specify the MQDeliveryMode, the message priority and the message lifetime.

```
MQStatus MQSendMessageToDestination(
                    const MQProducerHandle     producerHandle,
                    const MQMessageHandle      messageHandle,
                    const MQDestinationHandle  destinationHandle)
```

This sends the message to the specified destination. The destinationHandle is still valid after the call.

```
MQStatus MQSendMessageToDestinationExt(
                      const MQProducerHandle     producerHandle,
                      const MQMessageHandle      messageHandle,
                      const MQDestiantionHandle  destinationHandle,
                      MQDeliveryMode             msgDeliveryMode,
                      MQInt8                     msgPriority,
                      MQInt64                    msgTimeToLive )
```

This is same as MQ_sendMessageToDestination except that it allows to specify the MQDeliveryMode, the message priority and the message lifetime.

```
MQStatus MQReceiveMessageNoWait( const MQConsumerHandle   consumerHandle,
                                 MQMessageHandle *        messageHandle)
```

If a message is pending for the consumer, then this call immediately returns with the message in the messageHandle. Otherwise, it immediately returns an error waits. The consumer must not be created by MQ_createAsyncMessageConsumer or MQ_createAsyncDurableMessageConsumer, this applys to other synchronous message receiving methods.

```
MQStatus MQReceiveMessageWait( const MQConsumerHandle   consumerHandle,
                               MQMessageHandle *        messageHandle)
```

This waits until the specified consumer receives a message and returns this message in messageHandle. If there is already a message pending for this consumer, then this call returns it immediately and does not block. If an exception occurs, such as the connection closing before a message arrives, then this call returns with an error.

```
MQStatus MQReceiveMessageWithTimeout(
                        const MQConsumerHandle   consumerHandle,
                        MQInt32                  timeoutMilliSeconds,
                        MQMessageHandle *        messageHandle)
```

This waits for up to timeoutMicroSeconds microseconds until the specified consumer receives a message and returns the message in the messageHandle. If there is already a message pending for this consumer, then this

call returns it immediately and does not block. If an exception occurs before a message arrives or the timeout expires, then this call returns with an error.

```
MQStatus MQSetMessageArrivedFunc(
                          const MQConsumerHandle      consumerHandle,
                          const MQMessageArrivedFunc  messageCallback,
                          void *                      callbackData)
```

(**private**) A client can use this method to receive notification when a message arrives for a specific **synchronous** message consumer. A MQMessageArrivedFunc can only be set for a message consumer created by MQCreateMessageConsumer and MQCreateDurableMessageConsumer. When a message arrives for the specified consumer, the messageCallback is passed the consumerHandle and the callbackData void* pointer that was passed in to MQSetMessageArrivedFunc. The messageCallback function should return quickly and should not attempt to receive/process the message that just arrived. It MUST NOT call any other MQ client library function.

```
MQStatus MQAcknowledgeMessages( const MQSessionHandle    sessionHandle
                                const MQMessageHandle    messageHandle )
```

This acknowledges the `message` and all other messages that had been received before it on the same session. Clients may individually acknowledge messages or they may choose to acknowledge messages in application defined groups (which is done by acknowledging the last received message in the group). Messages that have been received but not acknowledged may be redelivered to the consumer.

2.2.4Properties

```
MQStatus MQCreateProperties( MQPropertiesHandle *  propertiesHandle )
```

This creates a new properties object.

The following two methods actually represent eight methods each where <*TYPE*> is substituted by String, Bool, Int8, Int16, Int32, Int64, Float32, or Float64 respectively.

```
MQStatus MQSet<TYPE>Property(
                        const MQPropertiesHandle  propertiesHandle
                        ConstMQString             key,
                        MQ<TYPE>                  value)
```

This sets the property associated with key to value. For example,

MQSetInt32Property(MQ_BROKER_PORT_PROPERTY, 7676, propertiesHandle);

```
MQStatus MQGet<TYPE>Property(
                      const MQPropertiesHandle  propertiesHandle
                      ConstMQString             key,
                      MQ<TYPE> *                value)
```

This returns the value of the property with the given key. For example,

MQInt8 value;
MQGetInt8Property("<property-name>", &value, propertiesHandle);

```
MQStatus MQGetPropertyType(
                    const MQPropertiesHandle  propertiesHandle
                    ConstMQString             key,
                    MQType *                  propertyType)
```

This returns the type of the property defined by key. For example,

MQType type;
MQGetPropertyType("<property-name>", &type, propertiesHandle);

```
MQStatus MQPropertiesKeyIterationStart(
                              const MQPropertiesHandle  propertiesHandle )
```

This starts an iteration through the property keys. It also resets an existing key iteration on the properties object. You cannot have multiple active iterations on the same properties object. Adding or removing properties to/from the properties object invalidates the iteration. This is used with MQPropertiesKeyIterationHasNext and MQPropertiesKeyIterationGetNext to iterate through all of the property keys. The property key interation methods are not MT-safe. The result is undefined if the properties that the propertiesHandle represents are updated while interation is in progress.

```
MQBool MQPropertiesKeyIterationHasNext(
                              const MQPropertiesHandle  propertiesHandle )
```

This returns MQ_TRUE if there are additional property keys in the iteration, and MQ_FALSE if there are no more property keys in the iteration. MQPropertiesKeyIterationStart must be called before this function.

```
MQStatus MQPropertiesKeyIterationGetNext(
                              const MQPropertiesHandle   propertiesHandle
                              ConstMQString *            key)
```

This retrieves the next key from the key iteration. Caller should not modify or free the key that is returned.

2.2.5Messages

```
MQStatus MQCreateTextMessage( MQMessageHandle *   messageHandle )
```

This creates a new text message.

```
MQStatus MQCreateBytesMessage( MQMessageHandle *  messageHandle )
```

This creates a new bytes message.

```
MQStatus MQCreateMessage( MQMessageHandle *  messageHandle )
```

This creates a new MQ_MESSAGE type message.

```
MQStatus MQSetMessageHeaders( const MQMessageHandle     messageHandle,
                               MQPropertiesHandle       headersHandle )
```

This sets the message headers for the message. Header fields not specified in headersHandle are not affected -- only the header fields specified by headersHandle are changed. The headersHandle is invalid after calling this function. The supported header properties, as shown below, are defined in mqheader-props.h. Unsuported properties in the headersHandle are ignored.

KEY	TYPE	Set By
MQ_PERSISTENT_HEADER_PROPERTY	MQBool	Send method*
MQ_EXPIRATION_HEADER_PROPERTY	MQInt64	Send method*
MQ_PRIORITY_HEADER_PROPERTY	MQInt8	Send method*
MQ_TIMESTAMP_HEADER_PROPERTY	MQInt64	Send method*
MQ_MESSAGE_ID_HEADER_PROPERTY	MQString	Send method*
MQ_CORRELATION_ID_HEADER_PROPERTY	MQString	client app

MQ_MESSAGE_TYPE_HEADER_PROPERTY	MQString	client app
MQ_REDELIVERED_HEADER_PROPERTY	MQBool	Message Broker**

*:When the message is sent, the value in the header is ignored; after completion of send, it holds the value that the sending method set it to. Therefore, generally these headers are used for MQGetMessageHeaders.
**: This header has no meaning on send.

```
MQStatus MQSetMessageProperties( const MQMessageHandle     messageHandle,
                                 MQPropertiesHandle        propertiesHandle)
```

This sets the message properties for the message. This function replaces all properties of the message with the properties specified in the propertiesHandle. The propertiesHandle is invalid after calling this function.

```
MQStatus MQSetTextMessageText( const MQMessageHandle  messageHandle,
                               ConstMQString          messageText )
```

This sets the text message's text. The messageHandle must have been created by calling MQ_createTextMessage. A copy of the messageText is made, so the caller can manipulate messageText after this function returns.

```
MQStatus MQSetBytesMessageBytes( const MQMessageHandle  messageHandle,
                                 const MQInt8 *         messageBytes,
                                 MQInt32                messageBytesSize )
```

This sets the bytes message's bytes. The messageHandle must have been created by calling MQCreateBytesMessage. A copy of the messageBytes is made, so the caller can manipulate messageBytes after this function returns.

```
MQStatus MQSetMessageReplyTo( const MQMessageHandle       messageHandle,
                              const MQDestinationHandle   destinationHandle )
```

This sets the reply to destination for this message. The destinationHandle is still valid after calling this function.

```
MQStatus MQGetMessageHeaders( const MQMessageHandle       messageHandle,
                              MQPropertiesHandle *        headersHandle )
```

This gets the message's headers. The caller is responsible for freeing messageHeaders by calling MQFreeProperties.

```
MQStatus MQGetMessageProperties( const MQMessageHandle       messageHandle,
                                 MQPropertiesHandle *        propertiesHandle )
```

This gets the message's properties. The caller is responsible for freeing propertiesHandle by calling MQFreeProperties.

```
MQStatus MQGetMessageType( const MQMessageHandle   messageHandle,
                           MQMessageType *         messageType )
```

This gets the type of the message. Currently, only MQ_TEXT_MESSAGE and MQ_BYTES_MESSAGE are supported.

```
MQStatus MQGetTextMessageText( const MQMessageHandle   messageHandle,
                               ConstMQString *         messageString )
```

This gets the text message's text. The type of the message referred to by messageHandle must be MQTextMessageType. The MQString that is returned is not a copy. The caller should not modify the bytes or attempt to free it.

```
MQStatus MQGetBytesMessageBytes( const MQMessageHandle  messageHandle,
                                 const MQInt8 **        messageBytes,
                                 MQInt32 *              messageBytesSize)
```

This gets the bytes message's bytes. The type of the message referred to by messageHandle must be MQBytesMessageType. The messageBytes that are returned is not a copy. The caller should not modify the bytes or attempt to free it.

```
MQStatus MQGetMessageReplyTo( const MQMessageHandle       messageHandle,
                              MQDestinationHandle *       destinationHandle )
```

This gets the reply to destination for this message. The caller is responsible for freeing destinationHandle by calling MQFreeDestination.

2.2.6Destination

```
MQStatus MQGetDestinationType( const MQDestinationHandle  destinationHandle,
                               MQDestinationType *        destinationType )
```

This gets the destination type of the destinationHandle.

```
MQStatus MQGetDestinationName( const MQDestinationHandle  destinationHandle,
                               MQString *  destinationName )
```

This gets the destination name of the destinationHandle.

2.2.7Errors

mqerrors.h will contain all the MQ C-API defined error codes and their descriptions will be in the MQ C-API documentation.

```
MQBool MQStatusIsError( const MQStatus  status )
```

This returns MQ_TRUE if only if status represents an error and MQ_FALSE otherwise.

```
MQError MQGetStatusCode( const MQStatus    status)
```

Returns the 32 bit error code associated with the status.

```
MQString MQGetStatusString( const MQStatus   status )
```

This converts the status to a MQString representation. The caller is responsible for freeing the returned MQString by calling MQ_freeStatusString.

```
MQString MQGetErrorTrace()
```

This will return the current thread's current error trace in string format as following or NULL if no error trace available. A MQ C-API internal function decides when or whether to record in the error trace in case of an error. Error trace strings are thread private.

```
method:__FILE__:__LINE__:error-type:error-code[:error-string]
......
```

where `error-type` like mq, nspr, nss, os.

The caller will be responsible for freeing the returned MQString by calling MQFreeString. The current thread's error trace will be automatically cleared on the next MQ C-API method call and in MQCloseConnection call. Note that the error trace string format is **private** – no guarantee compatibility in releases – and to be documented for debugging analysis.

2.2.8Logging

MQ logging facility aims for development and debugging purpose. Therefore all logging interfaces are private (no compatibility guarantee in releases) and will be documented selectively. For initial release, all logging levels will be compiled in and can be enabled for both optimized and debug builds.

MQ C-API library will use 2 environment variables to control execution-time logging:

MQ_LOG_FILE This environment variable specifies the file to which log messages are directed. If it's not defined, stderr will be used. By default, the actual log file name will be suffixed with .<n> where n is 0, 1, 2, ... that are rotation indices when max log file size is reached. A '%g' can be used to specify rotation index replacement in MQ_LOG_FILE after the last directory separator. Only the last '%g' will be replaced for rotation index if multiple '%g' specified. The '%g' replacement can be escape with '%'. If MQ_LOG_FILE is the name of a directory, it should include a trailing directory separator. The max rotation index is 9 and the max log file size is 1 MB, which are not configurable in the initial release.

MQ_LOG_LEVEL This is a numeric value as specified in MQLoggingLevel. When it sets to -1 (i.e. MQ_LOG_OFF), nothing will be logged. If it's not defined, both optimized and debug MQ builds have logging level set to 3 (i.e. MQ_LOG_CONFIG).

MQ_LOG_MASK (**private**) This is a numeric value to allow only selected areas in the C-API library to be logged

All the following logging methods are private and any of them may potentially be removed in future releases.

```
MQStatus MQSetLoggingFunc( MQLoggingFunc  loggingFunc,
                           const void *   callbackData )
```
This installs the user-defined callback function loggingFunc as the default logging function. The MQ client library logs a message by invoking this callback. The callbackData parameter will be passed to the logging function when each message is logged.

```
MQStatus MQSetLogFilePrefix( ConstMQString  loggingPrefix )
```
This sets the log file prefix name.

```
MQStatus MQSetMaxLogSize( MQInt32  maxLogSize )
```
This sets the log file maximum size. The actual size will slightly exceeds this because the log is closed whenever its size exceeds the maximum size. (note: not implemented yet)

```
MQStatus MQSetLogMask( MQLoggingLevel loglevel, MQInt32 logMask )
```
This sets the log mask for log level. This controls what messages actually get logged. (note: defined log masks not exported).

```
MQStatus MQGetLogMask( MQLoggingLevel loglevel, MQInt32 * logMask )
```
This gets the log mask for log level.

```
MQStatus MQSetCallbackLogLevel( MQLoggingLevel  logLevel )
```
This sets the logging level for the application installed logging callback function.

```
MQStatus MQSetLogFileLogLevel( MQLoggingLevel  logLevel )
```
This sets the log file logging level.

```
MQStatus MQSetStdErrLogLevel( MQLoggingLevel  logLevel )
```
This sets the stderr logging level.

```
MQStatus MQGetCallbackLogLevel( MQLoggingLevel * logLevel )
```
This gets the logging level for the application installed logging callback function.

```
MQStatus MQGetLogFileLogLevel( MQLoggingLevel * logLevel )
```
This gets the log file logging level.

```
MQStatus MQGetStdErrLogLevel( MQLoggingLevel * logLevel )
```
This gets the stderr logging level.

2.2.9Memory Deallocation

These methods are used to free memory allocated by the MQ client on behalf of the user. The methods MQCloseConnection, MQCloseSession, MQCloseMessageProducer, MQCloseMessageConsumer and MQFreeConnection are used to free resources associated with connections, sessions, producers, and consumers.

```
MQStatus MQFreeConnection( MQConnectionHandle connectionHandle )
```

This deletes all resources associated with the connection. This function can only be called after the connection was closed by calling MQ_closeConnection, and after all of the application threads associated with this connection and its decendant sessions, producers, consumers, etc. have returned. This function MUST NOT be called while an application thread is active in a library function associated with this connection or one of its decendant sessions, producers, or consumers. MQFreeConnection will release all resources held by the connection and its decendant sessions, producers, consumers, and destinations. It will not release resources held by messages associated with this connection. Resources held by a message must be explicitly freed by calling MQFreeMessage.

```
MQStatus MQFreeDestination( MQDestinationHandle destinationHandle )
```

This frees all memory associated with the destination specified by the destinationHandle. The client should not reference the destinationHandle after calling MQFreeDestination.

```
MQStatus MQFreeString( MQString  string )
```

This frees all memory associated with the string. The client should not reference the string after calling MQFreeString.

```
MQStatus MQFreeMessage( MQMessageHandle  messageHandle )
```

This frees all memory associated with message. The client should not reference the messageHandle after calling MQFreeMessage.

```
MQStatus MQFreeProperties( MQPropertiesHandle  propertiesHandle )
```

This frees all memory associated with properties. The client should not reference the propertiesHandle after calling MQFreeProperties.

2.3Public Headers

mqcrt.h includes all MQ C-API public headers:

mqtypes.h
mqbasictypes.h

mqconnection.h
mqsession.h
mqproducer.h
mqconsumer.h
mqdestination.h
mqmessage.h
mqbytes-message.h
mqtext-message.h
mqproperties.h
mqconnection-props.h
mqheader-props.h
mqcallback-types.h
mqssl.h
mqstatus.h
mqerrors.h
mqversion.h

2.3.1Preprocessor symbol names

A MQ C-API application should be compiled with -DSOLARIS, -DWIN32 (or /DWIN32), -DLINUX on platform Solaris, Windows, Linux respectively. These preprocessor micros are used in mqbasictypes.h to define MQ fixed-size integer types. mqbasictypes.h will automatically define these preprocessor symbol names for the supported compilers if it can identify them.

2.3.2mqversion.h

mqversion.h has following content, for example :

```
#define MQ_NAME      "Sun ONE Message Queue, Sun Microsystems, Inc."
#define MQ_VERSION "3.5"
#define MQ_VMAJOR    3
#define MQ_VMINOR    5
#define MQ_VMICRO    0
#define MQ_SVCPACK 0
```

3.MQ C-API X/Open XA Support

In 4.2.0, MQ C-API has added support for the X/Open XA Interface so that MQ C client applications running in a X/Open Distributed Transaction Processing (DTP) environment, such as BEA Tuxedo, can use MQ as a X/Open XA-compliant resource manager.

3.1 MQ C-API XA specific methods

In order to be able to participate XA transactions, a MQ C client application must use following MQ C-API funtions:

- **MQGetXAConnection** – to get the Connection handle
- **MQCreateXASession** – to create a Session

additionally for asynchronous message delivery (applications that use MQMessageListenerFunc), the application must pass both beforeMessageListener and afterMessageListener callback functions to

MQCreateXASession. Please see MQCreateXASession for details.

3.2MQ C-API XA Public Information

The following are MQ C-API XA public information required by the X/Open XA speccification:

- **The name the xa_switch_t structure**: sun_mq_xa_switch
- **The name of the RM**: SUN_RM
- **The xa_close string and format**: none
- **The MQ C-API library to be linked**: mqcrt
- **The xa_open string and format**: ';' separated name=value pairs

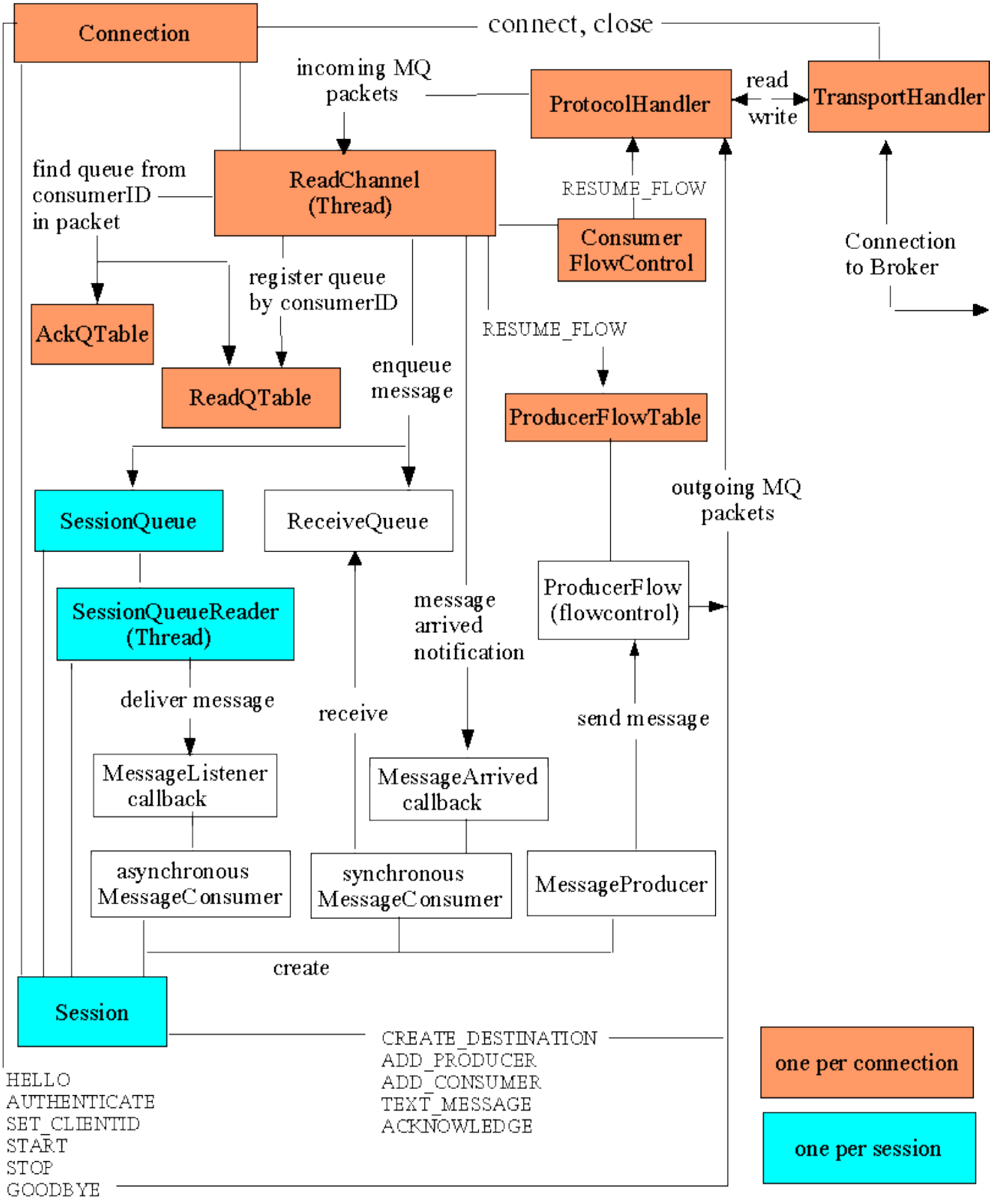
Name	*Value*	*Description*	*Default*
address	<host>:<port>	The host:port the broker's portmapper is running on	localhost:7676
username	string	The user name for the connection to broker	guest
password	string	The username's password	guest
conntype	TCP or SSL	The connection type to the broker	TCP
trustedhost	true or false	Whether the broker host is trusted (applicable only when conntype=SSL)	TRUE
certdbpath	string	The full path of the directory that contains NSS certificate and key database files	not set
clientid	string	Required only for JMS durable subscription application	not set
reconnects	integer	The number of reconnection attempts to broker (0 means no reconnect)	0

4.MQ C-API Runtime Architecture

4.1Design

The original design of the C-API runtime architecture was closely following the Java MQ client design. The current C-API runtime design has differences from the Java MQ client design in number of features that are

added for the initial release. The following diagram and table summarizes the current design.


Connection
connect, close
incoming MQ packets
ProtocolHandler
read
write
TransportHandler
find queue from consumerID in packet
ReadChannel (Thread)
RESUME_FLOW
Consumer FlowControl
Connection to Broker
register queue by consumerID
AckQTable
ReadQTable
enqueue message
RESUME_FLOW
ProducerFlowTable
SessionQueue
ReceiveQueue
outgoing MQ packets
SessionQueueReader (Thread)
message arrived notification
ProducerFlow (flowcontrol)
deliver message
receive
send message
MessageListener callback
MessageArrived callback
asynchronous MessageConsumer
synchronous MessageConsumer
MessageProducer
create
Session
CREATE_DESTINATION
ADD_PRODUCER
ADD_CONSUMER
TEXT_MESSAGE
ACKNOWLEDGE
HELLO
AUTHENTICATE
SET_CLIENTID
START
STOP
GOODBYE
one per connection
one per session

Component Name	Component Description
Connection	Connection handles connection level duties such as setting up the connection to the broker, authenticating to the broker, creating sessions, handling errors, and closing the connection to the broker.
Session	Session handles session level duties such as creating producers/consumers/destinations, ...
TransportHandler	TransportHandler implements the actual socket connection to the broker. TransportHandler can be either SSL or TCP, and will have a well-defined interface to allow future transports to be plugged in
ProtocolHandler	ProtocolHandler converts MQ messages to/from MQ wire protocol format.
ReadChannel	The ReadChannel thread dispatches incoming JMS messages to the appropriate message consumers. It uses ReadQTable to find the correct message consumer based on the consumerID stored in the packet.
Consumer FlowControl	The Consumer FlowControl handles flow control with the broker. It sends RESUME_FLOW messages to the broker once the client has sufficiently processed incoming messages. (Note: There is currently no flowcontrol thread in the implementation)
ProducerFlow	The ProducerFlow handles producer-based flow control to broker. When the ReadChannel receives RESUME_FLOW packet from broker for a particular producer, it resumes the corresponding ProducerFlow.
ReadQTable	The ReadQTable maps a consumerID to a ReceiveQueue or SessionQueue. When a message consumer is created, an unique consumerID is assigned to it. This consumerID is in each message packet that is delivering to this consumer, and ReadChannel uses it to demultiplex incoming messages to the correct queue.
AckQTable	The AckQTable is similar as ReadQTable but it's for client/broker control messages
ReceiveQueue	The ReceiveQueue maintains a queue of packets have been received but not yet delivered to the message consumer.
SessionQueue	A SessionQueue is associated with each Session that is created with asynchronous receive-mode
SessionQueue Reader	A SessionQueueReader thread is associated with each Session that is created with asynchronous receive-mode.
MessageListener Callback	The MesasgeListenerCallback is a callback function that installed by the MQ client application for a asynchronous message consumer. When a message for the consumer has arrived, the SessionQueueReader thread delivers the message to this function for processing..
MessageArrived Callback	The MesssageArrivedCallback is a callback function installed by the MQ client application for a synchronous message consumer. The ReadChannelcalls this function to notify the consumer that a message has arrived. The message consumer should not attempt to receive or process the message in this callback.
MessageConsumer	A sychronous MessageConsumer receives messages synchronously by calling MQ ReceiveMessage functions; A asynchronous MessageConsumer receives messages asynchronously through its MessageListenerCallback function
MessageProducer	The MessageProducer sends messages by MQ_sendMessage.

4.2Implementation

The client implementation will be done almost entirely in C++ using NSPR for portable socket access and thread creation. Unportable C++ features, such as templates, namespaces, and exceptions, will be avoided. All strings will be stored internally in UTF-8 format. Initially, the library will only export the C interface described above.

5.Security

MQ C-API uses NSS to support SSL transport protocol between MQ C Client and MQ broker. NSS supports SSL v2, SSLv3 and TLS. MQ C-API also uses NSS for MD5 digest to support MQ digest authentication type (Note: The digest function currently used is non-public that need to be replaced with a public NSS API).

For initial release, MQ basic authentication type will not be supported.

6.Internationalization

6.1Error String

All errors will be assigned a unique error code. MQ C-API error codes and their descriptions will be published in the header mqerrors.h. An application that uses MQ C-API can provide internationalized error messages for MQ error codes by using its own internationlization scheme. MQ_getStatusString always returns the UTF-8 encoded error string for error code returned from MQ_getStatusCode.

Alternatively, MQ C-API library can use ICU, the Internal Components for Unicode, to internationalize the error strings through ICU resource bundles. This would introduce another private contract dependency and potentially inconsistency with user application's internationlization. It can be considered in future if necessary.

6.2MQString

MQString, as it is specified, is a null terminated UTF-8 encoded character string. Since ASCII text is also UTF-8 text, a ASCII character string can be used wherever a MQString is required. For other platform specific encodings, users of MQ C-API must do their own conversion to or from UTF-8, for example, by using ICU (a freeware) or other facilities available to them on their platforms. Addition of such conversion utility methods to the MQ C-API can be considered in future if necessary.

7.64-bit clean

The MQ C-API shared library initially will be delivered as a 32-bit library. The API sources should be portable between 32-bit and 64-bit application environments, that is, be 64-bit ready. Since MQ 3.6 it supports 64-bit for Solaris sparc and 3.6SP2 for Solaris x86.

8.Dependencies

- MQ Client/Broker Protocol
  The MQ C-API was initially developed based on MQ 2.0 (Falcon) Client/Broker protocol. This will be changed to use MQ 3.5 Client/Broker (Raptor) protocol.

- NSPR
- NSS
- Netscape Communicator for client certificate/key db management (TBD)

Additionally, Sun ONE Directory Server will have a dependency on the MQ C-API.

9.Interfaces

Exported Interfaces:

Interface	Classification	Comments/Specification
MQ C-API • MQ C-API public headers • MQ C-API shared library	Evolving mqerrors.h: Evolving: common error codes Unstable: non-common error codes	this document see APPENDIX Error Codes for common errod codes
Environment Variables: MQ_LOG_FILE MQ_LOG_LEVEL	Unstable	this document
Package Names (Solaris) SUNWiqcrt (runtime) SUNWiqcdv (dev)	Unstable	MQ 3.5 Solaris Packaging Spec.

Imported Interfaces:

Interface	Classification	Comments/Specification
MQ Client/Broker Protocol	Commited Private	WS_MQ_RAPTOR/specs/protocol
NSPR	Contracted Consolidation Private	WSARC/2002/217
NSS	Contracted Consolidation Private	WSARC/2002/366

10.Performance

MQ C-API will be benchmarked by porting MQ JMSMarks, additionally will be measured by any existing performance program that was from Sun ONE Directory Server group. New APIs introduced for the initial release would add more processing in the library which can potentially affect performance. Equivalent JMSMarks benchmarking will also be done for selected compatitors C-API. Currently there is no base-line performance numbers available.

11.Build Environments

The build environments for the MQ C-API shared library are as following. The dependencies, NSPR and NSS, versions will be the latest RTM versions and in consistent with Orion. Linux version (including kernel version) may change depending on final versions of NSPR and NSS to be used and compiler versions may need to adjust accordingly.

platform	Build OS	Compiler	Compiler/linker options	NSPR	NSS

Solaris sparc	SunOS 5.8	Forte C++ compiler update 2	-compat=5 -O -mt -D_REENTRANT -KPIC -xregs=no%appl -DXP_UNIX -DSYSV -DSOLARIS -G -M libmqcrt.mapfile -z text -z defs -i -norunpath -B eliminate -L..... -lssl3 -lnss3 -lfreebl -lplc4 -lplds4 -lnspr4 -lpthread -ldl -lCrun -lc -R \$$ORIGIN:\$$ORIGIN/mps:\$$ORIGIN/../.. /lib:/usr/lib/mps	4.1.2	3.3.2
Windows	2000 (SP2)	Microsoft visual C++ 6.0 (SP4)	/nologo /MD /O2 /Ox /W3 /GT -DWIN32 -D_MT -D_DLL -D_WINDOWS -D_X86_ -DXP_WIN32 -DXP_WIN -DXP_PC -DMQ_EXPORT_DLL_SYMBOLS -DMQ_LIBRARY /FIXED:NO /WARN:3 /NODEFAULTLIB kernel32.lib msvcrt.lib /LIBPATH..... ssl3.lib nss3.lib freebl.lib libplc4.lib libplds4.lib libnspr4.lib	4.1.5	3.3.5
Linux	RedHat 7.2	gcc/g++ 2.96	-O2 -fPIC -D_REENTRANT -DXP_UNIX -DLINUX -W -L..... -lssl3 -lnss3 -lfreebl -lplc4 -lplds4 -lnspr4 -lpthread -ldl -Wl,-rpath\$ORIGIN: \$ORIGIN/mps:\$ORIGIN/../../lib	4.1.5	3.3.5
Solaris i386	SunOS 5.9 update 2 or later	Forte C++ compiler update 2	-compat=5 -O -mt -D_REENTRANT -KPIC -DXP_UNIX -DSYSV -DSOLARIS -G -M libmqcrt.mapfile -z text -z defs -i -norunpath -B eliminate -L..... -lssl3 -lnss3 -lfreebl -lplc4 -lplds4 -lnspr4 -lpthread -ldl -lCrun -lc -R \$$ORIGIN:\$$ORIGIN/mps:\$$ORIGIN/../.. /lib:/usr/lib/mps	4.1.3	3.3.3

11.1 mqcrt.so relocation requirment validation on Solaris (sparc)

ldd -r libmqcrt.so.1

```
        libssl3.so => /export/ws/binary/solaris/opt/lib/mps/libssl3.so
        libnss3.so => /export/ws/binary/solaris/opt/lib/mps/libnss3.so
        libplc4.so => /export/ws/binary/solaris/opt/lib/mps/libplc4.so
        libplds4.so => /export/ws/binary/solaris/opt/lib/mps/libplds4.so
        libnspr4.so => /export/ws/binary/solaris/opt/lib/mps/libnspr4.so
        libdl.so.1 =>    /usr/lib/libdl.so.1
        libCrun.so.1 =>  /usr/lib/libCrun.so.1
        libc.so.1 =>     /usr/lib/libc.so.1
        libpthread.so.1 =>       /usr/lib/libpthread.so.1
        libthread.so.1 =>        /usr/lib/libthread.so.1
        libposix4.so.1 =>        /usr/lib/libposix4.so.1
        libsocket.so.1 =>        /usr/lib/libsocket.so.1
        libnsl.so.1 =>   /usr/lib/libnsl.so.1
        libw.so.1 =>     /usr/lib/libw.so.1
        libaio.so.1 =>   /usr/lib/libaio.so.1
        libmp.so.2 =>    /usr/lib/libmp.so.2
        /usr/platform/SUNW,Ultra-2/lib/libc_psr.so.1
```

11.2MQ C-API Library Versioning (Solaris)

MQ C-API library on Solaris will be versioned and scoped by using Solaris versioning technology.

12. Supported compilers

Due to compatibility issues with C++ support libraries and lack of standards for naming mangling, virtual function implementation and class layout, the safest constraint is to require that all C++ code that executes in the same process to be compiled with the same compiler.

MQ C-API, being implemented in C++, requires compatible C++ runtime library available at runtime

Solaris: Sun WorkShop (Forte) 6 (update 2 or later) C++ Compiler with -compat=5
Sun C++ Compiler requirement:
The compiler that is used in the final link must be the latest compiler version in the mix.

Sun WorkShop C Compiler
mqcrt.so requires compatible libCrun at runtime
(Sun WorkShop 6 and newer versions C++ libraries are compatible)

Windows: Microsoft Windows Visual C++ 6.0 (SP3)
Linux: gcc/g++ 2.96

13.Supported platforms

Solaris 8 SPARC Solaris 9 SPARC Solaris 9 i386 (update 2 or later)
Windows 2000 Windows XP
RedHat 7.2

Note: supported OS versions on each platforms depend on schedule and QA resources

```
     Linux support may be a late binding depend on schedule.
```

```
Sun Linux support TBD (depending on NSPR and NSS, QA resource)
```

14.Distribution Content and File layout

14.1Distribution Content

- MQ C-API public headers
- NSPR and NSS libraries
- MQ C-API shared library
  Unix: libmqcrt.so.1
  Windows: mqcrt.lib mqcrt1.dll
- Samples (in C)
- Documentation

14.2File Layout

Solaris:

MQ C-API shared library is platform dependent and unbundled. It will go under /opt/packagename/lib, e.g. /opt/SUNWimq/lib. The distribution content will likely be in a separate package SUNWmqcrt. NSPR and NSS libraries will come from SUNWpr and SUNWtls packages and potentially their patches respectively. SUNWpr and SUNWtls are contract private (ref. WSARC 2002/217 and 2002/366).

```
/opt/SUNWimq/lib          <==== libmqcrt.so symlink to libmqcrt.so.1
/opt/SUNWimq/include
/opt/SUNWimq/demo/C
```

Linux: similar as on Solaris. A separate RPM may be necessary. NSPR/NSS libraries will be under /opt/imq/lib (see MQ3.5 Linux RPM Functional Specification) - subject change if NSPR/NSS decides to use RPMs.

Windows:

```
IMQ_HOME\lib              <==== add NSPR, NSS libs
IMQ_HOME\include
IMQ_HOME\demo\C
```

where IMQ_HOME refers to the root MQ installation directory, e.g. C:\Program Files\Sun\MessageQueue3. MQ C-API will be installed as a MQ module the same way as other MQ 3.5 modules are installed on Windows.

15.Instrumentation/Diagnostics

Please see C Client API Errors and Logging sections.

16.Configuration/Administration

MQ C-API library uses following 2 environment variables to control runtime logging of the library,
MQ_LOG_FILE
MQ_LOG_LEVEL
Please see C Client API Logging section for detail.

17.Compatibility/Interoperability/Migration

MQ 3.5 C-API client will not work with old broker. Broker version compatibility check will be at port mapper as well as at protocol level on connection creation. MQ C-API application can use MQGetMetaData and version constants defined in mqversion.h to check C-API runtime library version against compile-time library version. A sample will be provided to do just that. A version compatibility check API method can be considered for future release (mainly for non-Solaris platforms, for Solaris platform see library versioning on Solaris section).

18.Documentation

The following documentation must be provided

- MQ C-API documentation
- MQ C-API Developer's Guide
- Samples documentation (can be in samples source files)

19.Installation and Packaging

Please refer to File Layout section and MQ 3.5 installation related documentations. MQ C-API should be able to be downloaded and installed as a separate component of the MQ product.

20.License

MQ C-API component requires enterprise edition licensed broker. This will be enforced through MQ Client/Broker HELLO protocol.

21.References

Java Message Service(TM) 1.0.2 API Specification http://java.sun.com/products/jms/javadoc-102a/index.html

Sun ONE MQ C-API Prototype:
http://jpgserv.red.iplanet.com/JMQFalcon/engineering/details/client/c_api/index.html

JMQ 2.0 Client Design WS_JMQ_FALCON/specs/misc/clientDesign.html

APPENDIX Error Codes

When checking MQ C-API function return status, an application should only reference MQ C-API defined common error codes in order to maintain maximum compatibility with future releases. MQ C-API defined common error codes are listed in this appendix.

Function	*Common Error Codes*	*Comment*
MQAcknowledgeMessages	MQ_SESSION_NOT_CLIENT_ACK_MODE MQ_MESSAGE_NOT_IN_SESSION MQ_CONCURRENT_ACCESS MQ_SESSION_CLOSED MQ_BROKER_CONNECTION_CLOSED	

 ©Sun Microsystems, Inc 2001

Function	*Common Error Codes*	*Comment*
MQCloseConnection	MQ_CONCURRENT_DEADLOCK MQ_ILLEGAL_CLOSE_XA_CONNECTION	If called from exception listener or a consumer's message listener; If the connection is a XA connection
MQCloseMessageConsumer	MQ_CONSUMER_NOT_IN_SESSION MQ_BROKER_CONNECTION_CLOSED	
MQCloseMessageProducer	MQ_PRODUCER_NOT_IN_SESSION	
MQCloseSession	MQ_CONCURRENT_DEADLOCK	If called from a consumer's message listener in the session
MQCommitSession	MQ_NOT_TRANSACTED_SESSION MQ_CONCURRENT_ACCESS MQ_SESSION_CLOSED MQ_BROKER_CONNECTION_CLOSED MQ_XA_SESSION_IN_PROGRESS	
MQRollbackSession	MQ_NOT_TRANSACTED_SESSION MQ_CONCURRENT_ACCESS MQ_SESSION_CLOSED MQ_BROKER_CONNECTION_CLOSED MQ_XA_SESSION_IN_PROGRESS	
MQCreateConnection	MQ_INCOMPATIBLE_LIBRARY MQ_CONNECTION_UNSUPPORTED_TRANSPORT MQ_COULD_NOT_CREATE_THREAD MQ_INVALID_CLIENTID MQ_CLIENTID_IN_USE MQ_SSL_NOT_INITIALIZED MQ_COULD_NOT_CONNECT_TO_BROKER MQ_BROKER_CONNECTION_CLOSED	MQ_SSL_NOT_INITIALIZED if MQ_CONNECTION_TYPE_PROPERTY is SSL
MQGetXAConnection	MQ_STATUS_INVALID_HANDLE	
MQCreateDestination	MQ_INVALID_DESTINATION_TYPE MQ_SESSION_CLOSED	
MQCreateDurableMessageConsumer	MQ_NOT_SYNC_RECEIVE_MODE MQ_INVALID_MESSAGE_SELECTOR MQ_DESTINATION_CONSUMER_LIMIT_EXCEEDED MQ_TEMPORARY_DESTINATION_NOT_IN_CONNECTION MQ_CONSUMER_NO_DURABLE_NAME MQ_QUEUE_CONSUMER_CANNOT_BE_DURABLE MQ_CONCURRENT_ACCESS MQ_SESSION_CLOSED MQ__BROKER_CONNECTION_CLOSED	

Function	*Common Error Codes*	*Comment*
MQCreateMessageConsumer	MQ_NOT_SYNC_RECEIVE_MODE MQ_INVALID_MESSAGE_SELECTOR MQ_DESTINATION_CONSUMER_LIMIT_EXCEEDED MQ_TEMPORARY_DESTINATION_NOT_IN_CONNECTION MQ_CONCURRENT_ACCESS MQ_SESSION_CLOSED MQ_BROKER_CONNECTION_CLOSED	
MQCreateAsyncDurableMessageConsumer	MQ_NOT_ASYNC_RECEIVE_MODE MQ_INVALID_MESSAGE_SELECTOR MQ_DESTINATION_CONSUMER_LIMIT_EXCEEDED MQ_TEMPORARY_DESTINATION_NOT_IN_CONNECTION MQ_CONSUMER_NO_DURABLE_NAME MQ_QUEUE_CONSUMER_CANNOT_BE_DURABLE MQ_CONCURRENT_ACCESS MQ_SESSION_CLOSED MQ_BROKER_CONNECTION_CLOSED	MQ_CONCURRENT_ACCESS if connection not in stopped mode
MQCreateAsyncMessageConsumer	MQ_NOT_ASYNC_RECEIVE_MODE MQ_INVALID_MESSAGE_SELECTOR MQ_DESTINATION_CONSUMER_LIMIT_EXCEEDED MQ_TEMPORARY_DESTINATION_NOT_IN_CONNECTION MQ_CONCURRENT_ACCESS MQ_SESSION_CLOSED MQ_BROKER_CONNECTION_CLOSED	MQ_CONCURRENT_ACCESS if connection not in stopped mode
MQCreateMessageProducer	MQ_SESSION_CLOSED	
MQCreateMessageProducerForDestination	MQ_SESSION_CLOSED MQ_BROKER_CONNECTION_CLOSED	
MQCreateSession	MQ_INVALID_ACKNOWLEDGE_MODE MQ_INVALID_RECEIVE_MODE MQ_BROKER_CONNECTION_CLOSED MQ_COULD_NOT_CREATE_THREAD	
MQCreateXASession	MQ_NOT_XA_CONNECTION MQ_INVALID_RECEIVE_MODE MQ_BROKER_CONNECTION_CLOSED MQ_COULD_NOT_THREAD	
MQCreateTemporaryDestination	MQ_INVALID_DESTINATION_TYPE MQ_SESSION_CLOSED	
MQFreeConnection	MQ_STATUS_CONNECTION_NOT_CLOSED	
MQGet<*TYPE*>Property	MQ_NOT_FOUND MQ_INVALID_TYPE_CONVERSION	
MQGetMessageProperties	MQ_NO_MESSAGE_PROPERTIES	
MQGetMessageReplyTo	MQ_NO_REPLY_TO_DESTINATION	
MQGetPropertyType	MQ_NOT_FOUND	

Function	*Common Error Codes*	*Comment*
MQInitializeSSL	MQ_INCOMPATIBLE_LIBRARY MQ_SSL_ALREADY_INITIALIZED	
MQReceiveMessageNoWait	MQ_NOT_SYNC_RECEIVE_MODE MQ_CONCURRENT_ACCESS MQ_NO_MESSAGE MQ_CONSUMER_CLOSED MQ_SESSION_CLOSED MQ_BROKER_CONNECTION_CLOSED	
MQReceiveMessageWait	MQ_NOT_SYNC_RECEIVE_MODE MQ_CONCURRENT_ACCESS MQ_CONSUMER_CLOSED MQ_SESSION_CLOSED MQ_BROKER_CONNECTION_CLOSED	
MQReceiveMessageWithTimeout	MQ_NOT_SYNC_RECEIVE_MODE MQ_CONCURRENT_ACCESS MQ_TIMEOUT_EXPIRED MQ_CONSUMER_CLOSED MQ_SESSION_CLOSED MQ_BROKER_CONNECTION_CLOSED	
MQRecoverSession	MQ_TRANSACTED_SESSION MQ_CONCURRENT_ACCESS MQ_SESSION_CLOSED MQ_BROKER_CONNECTION_CLOSED	
MQSendMessage	MQ_PRODUCER_NO_DESTINATION MQ_PRODUCER_CLOSED MQ_SESSION_CLOSED MQ_BROKER_CONNECTION_CLOSED	
MQSendMessageExt	MQ_PRODUCER_NO_DESTINATION MQ_INVALID_PRIORITY MQ_INVALID_DELIVERY_MODE MQ_PRODUCER_CLOSED MQ_SESSION_CLOSED MQ_BROKER_CONNECTION_CLOSED	
MQSendMessageToDestination	MQ_PRODUCER_HAS_DEFAULT_DESTINATION MQ_PRODUCER_CLOSED MQ_SESSION_CLOSED MQ_BROKER_CONNECTION_CLOSED	
MQSendMessageToDestinationExt	MQ_PRODUCER_HAS_DEFAULT_DESTINATION MQ_INVALID_PRIORITY MQ_INVALID_DELIVERY_MODE MQ_PRODUCER_CLOSED MQ_SESSION_CLOSED MQ_BROKER_CONNECTION_CLOSED	
MQSet<*TYPE*>Property	MQ_HASH_VALUE_ALREADY_EXISTS	
MQSetMessageHeaders	MQ_PROPERTY_WRONG_VALUE_TYPE	

Function	*Common Error Codes*	*Comment*
MQStartConnection	MQ_BROKER_CONNECTION_CLOSED	
MQStopConnection	MQ_BROKER_CONNECTION_CLOSED MQ_CONCURRENT_DEADLOCK	MQ_CONCURRENT_DEADLOCK if called from a consumer's message listener
MQUnsubscribeDurableMessageConsumer	MQ_CANNOT_UNSUBSCRIBE_ACTIVE_CONSUMER MQ_CONSUMER_NOT_FOUND	

In addition, the following common error codes may return from any MQ C-API function that returns MQStatus:

MQ_STATUS_INVALID_HANDLE

MQ_OUT_OF_MEMORY

MQ_NULL_PTR_ARG

and MQ_TIMEOUT_EXPIRED may return from MQ C-API functions that communicate with MQ broker if the connection MQ_ACK_TIMEOUT_PROPERTY is set to a non-zero value. All MQ C-API functions that return a MQStatus will return error code MQ_SUCCESS on success.

MQ_THREAD_OUTSIDE_XA_TRANSACTION or MQ_XA_SESSION_NO_TRANSATION may return from MQSendMessage* or MQReceiveMessage* functions on a XA session.

MQ_CALLBACK_RUNTIME_ERROR can be returned from beforeMessageListener of MQMessageListenerBAFunc callback.

Some MQ C-API functions may return a MQStatus that contains a NSPR or NSS library error code instead of a MQ error code when they fail. For NSPR and NSS library error code, MQGetStatusString() will return the symbolic name of the NSPR or NSS library error code. Please see NSPR and NSS public documentation for NSPR and NSS error code symbols and their interperatation:

http://www.mozilla.org/projects/nspr/reference/html/index.html
on Error Codes in 'NSPR Error Handling' chapter

http://www.mozilla.org/projects/security/pki/nss/ref/ssl/
on SSL Error Codes in 'NSS and SSL Error Codes' chapter
on SEC Error Codes in 'NSS and SSL Error Codes' chapter

 ©Sun Microsystems, Inc 2001

Revision History

Date	Version	Description	Author
09/FEB/2008		Added certdbpath to supported xa_open string	Amy Kang
07/NOV/2007	4.2.0	1) Added MQGetConnectionProperties 2) added isXA param to MQCreateConnectionExt 3) added section 3. MQ C-API X/Open XA Support 4) added MQGetXAConnection 5) added MQCreateXASession 6) added callback type MQMessageListenerBAFunc 7) added new error codes: MQ_THREAD_OUTSIDE_XA_TRANSACTION MQ_XA_SESSION_NO_TRANSATION MQ_XA_SESSION_IN_PROGRESS MQ_CALLBACK_RUNTIME_ERROR MQ_NOT_XA_CONNECTION MQ_ILLEGAL_CLOSE_XA_CONNECTION	Amy Kang
04/NOV/2005	1.0.1	8) Added support for MESSAGE type	Amy Kang
21/OCT/2005	1.0.1	9) Added MQGetDestinationName	Amy Kang
27/OCT/2003	1.0	10) Changed MQ_BROKER_NAME_PROPERTY to MQ_BROKER_HOST_PROPERTY	Amy Kang
13/OCT/2003	1.0	11) Explicitly specified that connection must be in stopped mode when create async message consumer	Amy Kang
11/OCT/2003	1.0	12) Added APPENDIX Error Codes 13) explicitly clarify error codes in Exported Interface table	Amy Kang
06/OCT/2003	1.0	14) Specified log file generation and %g rotation index replacement 15) add MQ_SVCPACK to mqversion.h 16) add automatically defining preprocessor symbols SOLARIS, LINUX and WIN32 for supported compilers 17) changed build environment for Windows to use MSVC++ SP4 18) specified that MQPropertiesKeyIterationStart will reset existing key iteration	Amy Kang
04/JUL/2003	1.0	19) Connection properties name changes for connection based consumer flowcontrol	Amy Kang

30/JUN/2003	1.0	20) Memory management semantic changes for destinationHandle: now it's calllers responsibility to free the destinationHandle after creation; the destinationHandle that is passed to the consumer/producer creation functions will be still valid after the function call 21) specified that it's the C-API client's responsibility to free the messageHandle that is passed to the MQMessageListenerFunc callback; removed 'const' for messageHandle parameter in the MQMessageListenerFunc function type 22) removed 'const' for callbackData parameter in MQSetMessageArrivedFunc to sync with current source 23) removed 'const' for propertiesHandle parameter in MQCreateConnection, MQSetMessageProperties and MQSetMessageHeaders, and specified that the propertiesHandle is invalid after the call 24) removed 'const' for exception parameter in MQConnectionExceptionListenerFunc type 25) removed doc directory in file layout 26) added Solaris 9 i386 to supported platforms	Amy Kang
05/JUN/2003	1.0	27) Updated MQ C-API Runtime Architecture description , design diagram and table to in-sync with current implementation	Amy Kang
05/JUN/2003	1.0	28) Clarified error trace string format 29) specified MQSetMessageArrivedFunc will be private and can only be used for synchronous message consumers 30) updated NSPR and NSS versions to 4.1.4 and 3.3.4 respectively, and specified that final NSPR and NSS versions to be used will be their latest RTM versions and in consistent with Orion	Amy Kang
28/MAY/2003	1.0	31) Specified that preprocessor symbol names SOLARIS, WIN32 or LINUX should be used to compile C-API applications 32) clarified supported transports for the initial release	Amy Kang
21/MAY/2003	1.0	33) Added void * messagelistenerCallbackData parameter in async-consumer creation functions; added void * callbackData to the corresponding MQMessageListenerFunc callback function type. 34) changed MQExceptionListener to MQConnectionExceptionListener 35) specified what flowcontrol features will be supported in the initial release 36) added MQ_LOG_MASK **private** env var	Amy Kang

07/MAY/2003	1.0	37) Specified that MQ integer types will be defined using sized-integer types on current supproted platforms, therefore removed NSPR headers from Distribution Content section 38) added mqbasictypes.h to Public Headers section 39) added type MQUint32 needed for MQError type 40) Changed samples directory to demo/C and removed CPP sample directory in File layout section 41) changed message header property MQ_MESSAGE_TYPE_HEADER_PROPERTY to be MQString type	Amy Kang
25/APR/2003	1.0	42) Removed MQStatusIsSuccess methods 43) corrected typo in description for MQCreateDurableMessageConsumer	Amy Kang
26/MAR/2003	1.0	44) Added support for AUTO and DUPS-OK acknowledge modes: MQAckMode enum now has values MQ_AUTO_ACKNOWLEDGE, MQ_DUPS_OK_ACKNOWLEDGE and SESSION_TRANSACTED besides MQ_CLIENT_ACKNOWLEDGE 45) added method MQGetAcknowledgeMode 46) removed exportPolicy parameter in MQInitializeSSL based on WSARC inception review 47) added MQ_NAME in mqversion.h and adjusted metadata property names for MQGetMetaData to be consistent with other property names	Amy Kang
07/FEB/2003		48) have MQMessageListenerFunc return MQError and added description of it.	Amy Kang
31/JAN/2003	1.0	49) removed private .h files in Public Headers section 50) added License section	Amy Kang
30/JAN/2003	1.0	51) added logging environment variables and Solaris package names to the exported interface table 52) updated section 1.2 to specify for initial release supported and not supported features	Amy Kang
27/JAN/2003	1.0	53) changed all methods to use function name style MQCreateConnection 54) removed from mqheader-props.h MQ_DESTINATION_HEADER_PROPERTY 55) clarified message headers for MQSet/GetMessageHeaders	Amy Kang

15/JAN/2003	1.0	added sections: Dependencies Interfaces Performance Compatibility/Interoperability Documentation	Amy Kang
14/JAN/2003	1.0	mostly from review feedback: 1) moved param propertiesHandle to first argument in property methods 2) changed public header file names to lower-case, drop 'Shim' in file names 3) specified not to invalid the propertiesHandle/ headersHandle on return from MQ_setMessageProperties MQ_setMessageHeaders 4) included supported message header props in description 5) changed MQ_freeStatusString to MQ_freeString 6) changed MQ_LOG_NONE to MQ_LOG_OFF to be consistent with JDK log level naming 7) add description for log levels 8) add description for MT and concurrent-update restrictions for properties key interation methods 9) consistent 'const' semantic for handle params to mean the handle's validity is not affected	Amy Kang
07/JAN/2003	1.0	mostly from review feedback: 1). changed MQDeliveryMode to enum type 2). added data type MQReceiveMode and param receiveMode to MQ_createSession 3). added param clientID to MQ_createConnection and removed connection property MQ_CONFIGURED_CLIENT_ID_PROPERTY 4). removed MQ_setExceptionListenerFunc and added corresponding parameters to MQ_createConnection 5). use enum MQDestinationType in MQ_createDestination 6). changed “With” to “For” in MQ_createMessageProducerWithDestination 7). changed “Ex” to “Ext” in method names 8). changed “sendMessageTo” to “sendMessageToDestination” in MQ_sendMessageTo 9). changed MQ_acknowledgeMessage to MQ_acknowledgeMessages (sessionHandle , messageHandle) 10).use milliseconds for timeout param in MQ_receiveMessageWithTimeout 11).added library versioning section for Solaris	Amy Kang

20/DEC/2002	1.0	1). changed enum data types' values to all CAP 2). removed MQ_setCreateThreadFunc 3). changed following data types or functions to **private**: MQThreadFunc MQCreateThreadFunc MQ_setMessageArrivedFunc MQMessageArrivedFunc all logging related data types and methods 4). added MQ_createConnectionEx which takes customerized thread creation function pointer 5). clarified property related functions 6). separated logging section and added environment variables to control C-API library runtime logging 7). added MQError data type and public header mqerrors.h 8). introduced error trace in Errors and added API MQ_getErrorTrace 9). added following data type and API method for asynchronous message receiving MQMessageListenerFunc MQ_createAsyncMessageConsumer MQ_createAsyncDurableMessageConsumer 10).modified I18n section to specify leave error string i18n to applications	Amy Kang
12/DEC/2002	1.0	1). iMQ to MQ name changes 2). removed const from function params like TYPE * const p const MQInt32 value 3). specify support for selector transaction (JMS) session recover 4). add following API methods MQ_getDestinationType MQ_commitSession MQ_rollbackSession MQ_recoverSession MQ_getMetaData 5). add data type MQDestionationType 6). add public header MQVersion.h 7). add following sections Public Headers Security 64-bit clean Build Environments Supported Compilers Supported Platforms Distribution content and File layout 8). changed consumer creation APIs for selector 9). note to add FlowContorl thread	Amy Kang

22/NOV/2002	.15	Update for domain unification, data types, API function decriptions, iMQ to MQ name changes in text area. Functions, data types and design diagram sync'ed with source	Amy Kang
14/JAN/2002	.14	Renamed classes to match Java implementation	David Ely
9/DEC/2001	.13	Added point-to-point methods, synchronous receive, thread creation callback, and message arrived callback. Removed asynchronous receive.	David Ely
16/NOV/2001	.12	Added section on internationalization. Removed string conversion routines.	David Ely
13/NOV/2001	.11	Applied comments from iMQ group. Extended handling of properties.	David Ely
6/NOV/2001	.1	Initial Draft	David Ely

©Sun Microsystems, Inc 2001