MA0S

WebSphere MQ – JMS API Exerciser & Code Generator Version 1.0

12 Dec 2002

Larry Yusuf

Hursley Solution Test Centre

WebSphere Platform System House

yusuf@uk.ibm.com

Table of Contents

Notices

Trademarks and service marks

Acknowledgement

I would like to thank Mick Lickman for all the guidance provided for WebSphere MQ, Kareem Yusuf for his assistance in developing the GUI and for reviewing parts of the design, and Clare Jackson for reviewing this User Guide.

Summary of Amendments

Date	Changes
01 December 2002	Initial Release

Preface

The JMS API Exerciser & Code Generator is a simple and convenient tool that assists designers, application developers, and testers of Messaging solutions, by allowing WebSphere MQ Java Messaging Service (JMS) functionality to be explored independently of the actual application.

The SupportPac is a simple generic Point-to-Point client that allows potential solutions based on WebSphere MQ (using JMS) to be investigated. The SupportPac provides a code generation facility that shows the interface, classes, and methods involved in achieving the tasks being performed. This code generation facility allows users to familiarize themselves with the JMS API and writing JMS applications.

This document introduces the JMS API Exerciser & Code Generator, describes the required installation and configuration steps, presents the functions, menus, and screen shots, to enable the effective and efficient use of the tool.

Although no official support is provided, the author is interested in hearing of any problems or suggestions for improvement to this SupportPac. If a bug is suspected, please send an email with a problem description. The author's email address is on the front cover of this document.

Bibliography

- MQSeries Using Java, IBM Corporation. SC34-5456-07
- Java Message Service, O’Reilly UK. 0596000685
- WebSphere MQ Integrator, Programming Guide v2.1, IBM Corporation, SC34-5603-03
- JMS API, http://java.sun.com/j2ee/sdk_1.3/techdocs/api/

Chapter
1

Introduction

The Point-to-Point Messaging Paradigm allows messaging clients to send and receive messages both synchronously and asynchronously via virtual channels known as queues. These messaging clients (in this case, JMS Client) interact with a Messaging System / Message-Oriented Middleware (called the JMS Provider). WebSphere MQ is a functionally-rich Message-Oriented Middleware (MOM) that assures the delivery of messages to/from multiple producers and consumers.

Currently, there are utilities available that use WebSphere MQ Java API to PUT messages on queues while others use the C API to PUT messages to and GET messages from queues. There is no platform-independent utility that allows all Point-to-Point tasks to be performed simply, using JMS, or indeed that introduces the JMS API to Java programmers and others who work in the messaging domain.

This JMS API Exerciser & Code Generator SupportPac provides:

- A simple and convenient tool that allows Point-to-Point functionality of WebSphere MQ to be explored and demonstrated without having to invest time developing client applications (such as Message Producers and Consumers).
- A code snippet generation facility that allows those unfamiliar with the JMS API to acquaint themselves with the required classes and methods to perform specific operations.
- Support for multiple clients and connections and as such allows the registration of multiple connections, senders, receivers, and browsers from a simple GUI interface.
- Application development support by allowing the user to PUT messages on to queues, GET messages from queues and BROWSE the contents of queues.
- Helps the user to understand how the JMS API is being used to accomplish the different tasks being performed, by generating code snippets for each function performed whilst referencing the JMS API.

Since the JMS API Exerciser & Code Generator SupportPac is a pure Java application with no built-in platform dependencies, it is positioned to work on any platform that provides a suitable JVM. The SupportPac uses only the JMS API and has been tested with the WebSphere MQ JMS Implementation.

Chapter
2

Installation and Setup

Packaging

This SupportPac is composed of the following files:

■ MA0S.pdf	This User Guide
■ MA0S.cmd	Command file to launch the utility on the Windows platform
■ com.ibm.MA0S.jar	Contains application files
■ MA0S.dat	Default configuration file

To install, unzip the ma0s.zip distribution file into an executable directory (for example, c:\tools\MA0S).

Prerequisites

This MA0S SupportPac is a pure Java application that has no built-in platform dependencies, and as such, should be able to run on any platform that provides a suitable JVM and supports all other prerequisites. The following are required:

- Java 2 SDK, Standard Edition version 1.3.1, or higher
- WebSphere MQ version 5.2 or higher
- SupportPac MA88 (only required if the MQ version in use is earlier than version 5.3)

Associated product documentation describes the installation and configuration of the prerequisites.

Chapter
3

MA0S Tour

This chapter describes:

- The Configuration of the MA0S SupportPac environment
- The JMSQueueSender Tab for sending messages
- The RFH2/Usr Tab for creating RFH2 headers and user defined properties
- The JMSQueueReceiver/QueueBrowser Tab for accessing messages on queues.
- The JMS API Tab for displaying the generated code
- The tools menu options

Configuration

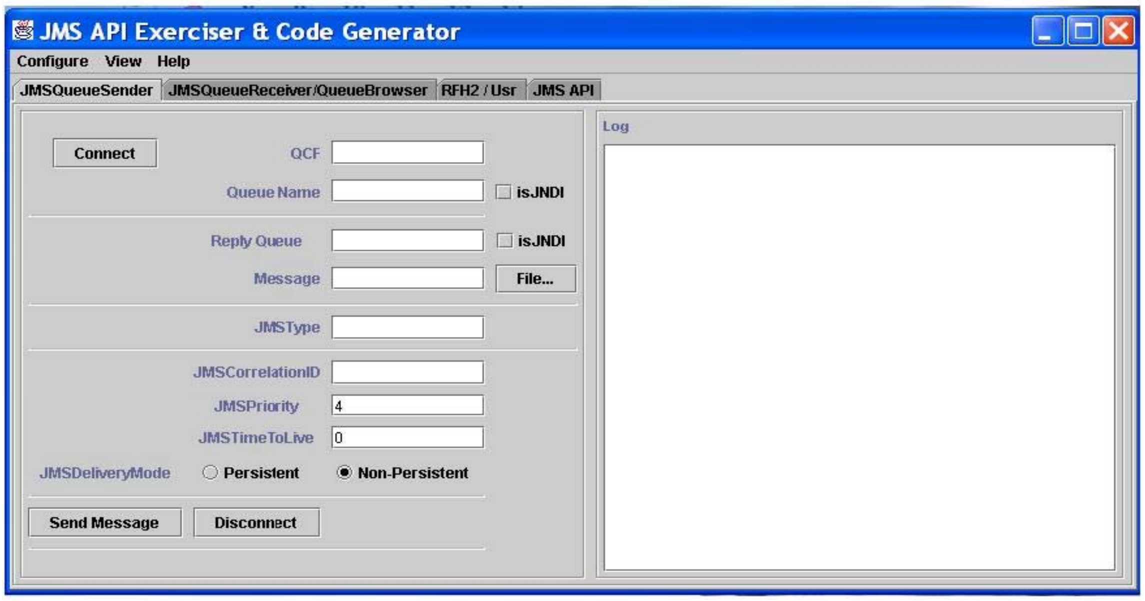
The MA0S.cmd file holds the commands required to start the tool, and provides a shortcut on the Windows platform. It contains the following:

- *SET PATH=,;%PATH%*
- *SET CLASSPATH=.;com.ibm.MA0S.jar;%CLASSPATH%*
- *java com.ibm.MA0S.MA0S*

The *SET PATH* command sets the runtime path to the system path. The directory hosting the java runtime (java.exe) must be on the path, for example, C:\Programs\Java131\bin.

The *SET CLASSPATH* command sets the runtime classpath to the system classpath plus the current directory and the application jars. It is assumed that the system classpath has been appropriately setup for SupportPac MA88, if used. If the MA0S.cmd file is not being used (for instance, on a UNIX-based system), the MA0S jar must be added to the system or runtime environment classpath.

java com.ibm.MA0S.MA0S runs the main executable MA0S.class resulting in the screen shown Figure 1.



JMSPriority: 4
JMSTimeToLive: 0
JMSDeliveryMode: Persistent [✓ Non-Persistent]

Figure 1: The initial screen loaded when the MA0S.cmd file is executed.

Configuring the tools environment

Select the *Configure→Environment...* menu. The resulting dialog box shown in figure 2, allows the user to specify the configuration options for JMS. The configuration options are summarized by the following variables:

- **Initial Context Factory**: This identifies the type of namespace being used and typically has the same value as that set in the *JMSAdmin.config* file used by SupportPac MA88 or the Integrated JMS implementation of WebSphere MQ v 5.3.

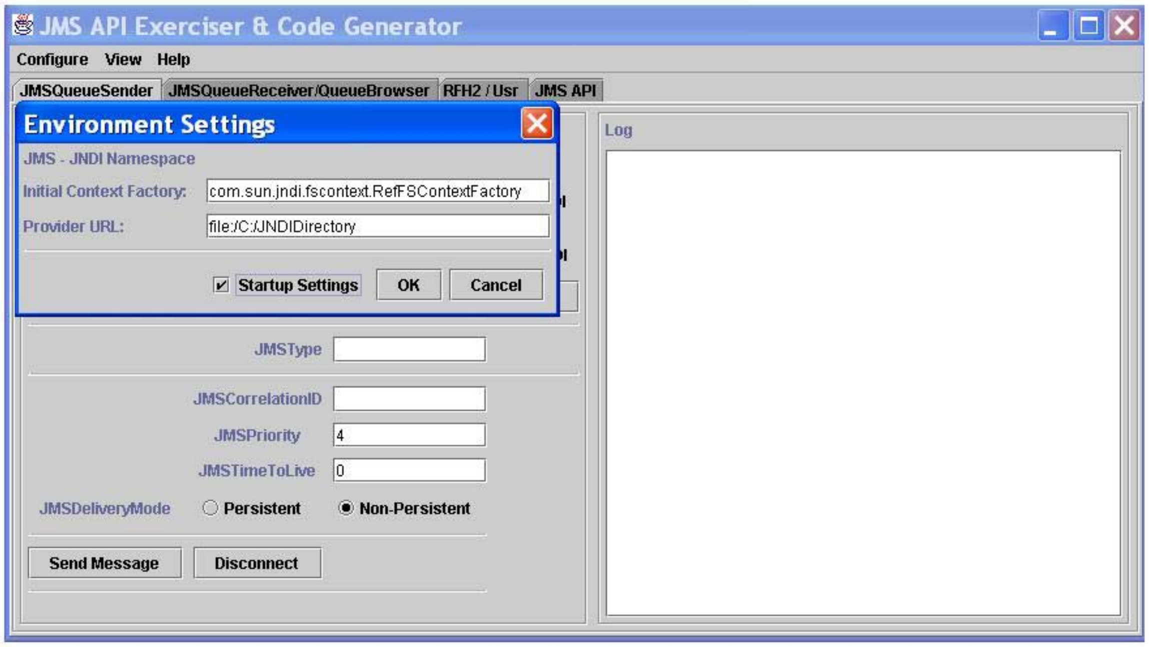
- **Provider URL**: This defines the location of the namespace and typically has the same value as that set in the *JMSAdmin.config* file used by the SupportPac MA88 or the Integrated JMS implementation of WebSphere MQ version 5.3.

If the "startup settings" checkbox is checked, the configuration is persisted to a file (MA0s.dat) in the install directory. This allows the configuration to be restored when the utility is restarted. It is important to note that the default configuration file that ships with the SupportPac assumes that:

1) The Initial Context Factory is com.sun.jndi.fscontext.RefFSContextFactory

2) The Provider URL is file:/C:/JNDIDirectory.

If the JNDI Namespace presented above does not exist, or has not been setup as required, an "Unable to Setup JNDI Namespace" error message is displayed in the log board.



Environment Settings

JMS - JNDI Namespace

Initial Context Factory: com.sun.jndi.fscontext.RefFSContextFactory

Provider URL: file:/C:/JNDIDirectory

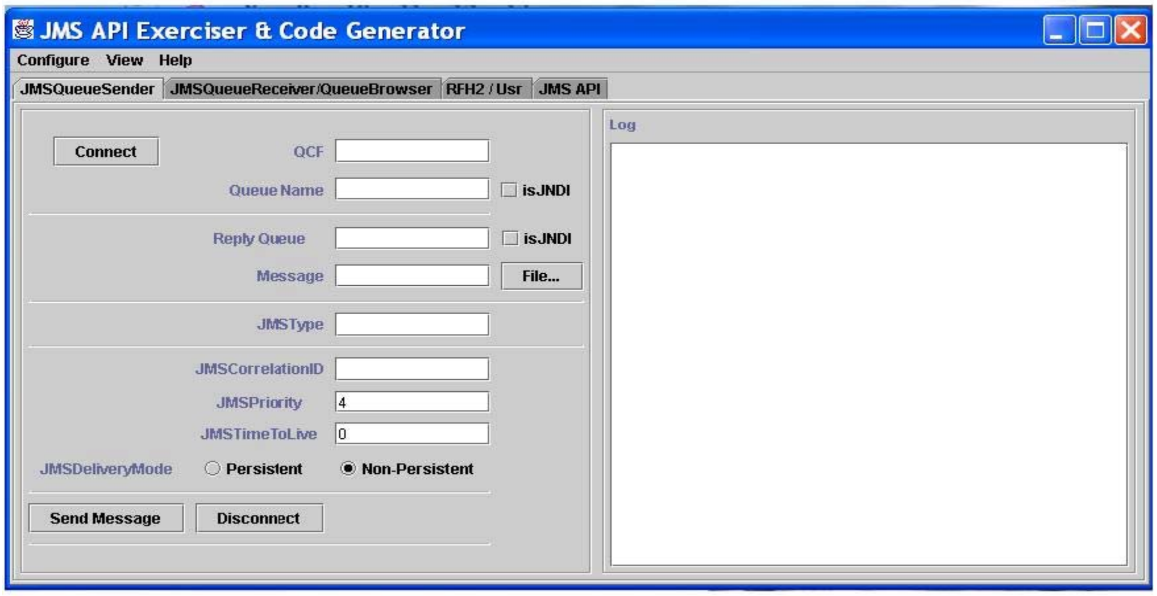
☑ Startup Settings | OK | Cancel

JMSType:
JMSCorrelationID:
JMSPriority: 4
JMSTimeToLive: 0
JMSDeliveryMode: ○ Persistent [✓ ● Non-Persistent]

Figure 2: The Environment Dialog box for the MA0S SupportPac

The JMSQueueSender Tab

The JMSQueueSender Tab shown in Figure 3 provides access to the message producing function of the tool. Interaction is based on three commands; Connect, Send Message, and Disconnect.



QCF:
Queue Name: ☐ isJNDI
Reply Queue: ☐ isJNDI
Message:
JMSType:
JMSCorrelationID:
JMSPriority: 4
JMSTimeToLive: 0
JMSDeliveryMode: ○ Persistent [✓ ● Non-Persistent]

Figure 3: The JMSQueueSender Tab

Connect Command

Connect is the first operation to be performed for the tool to send messages to a queue.

1) To connect, the name of the *QueueConnectionFactory* (QCF) to be used must be typed in the QCF field. The status of the connection (started or exception) is displayed in the log window.

2) For multiple connections, simply type the name of another QCF and connect.

3) Before issuing any other command, ensure that the desired QCF is typed / displayed in the QCF text field.

Send Message Command

The *send* message command takes the following fields as argument:

- **Queue Name**: The queue name where the message is sent must be supplied. The queue can be either a queue object to be retrieved from the JNDI namespace, or an actual queue string. This is in keeping with the JMS API that offers both options. If a queue object is supplied, the ***isJNDI*** checkbox should be selected, otherwise reference to a queue string is assumed. If a queue string is supplied, the queue name must be preceded by a forward slash "/", that is, ***/TestQ*** or ***/TestQMgr/TestQ***.

- **Reply Queue**: The reply queue name is the name of the queue where all reply messages to the message being sent should be put. This is similar to the queue name field above, in that the reply queue can be a queue object or a queue string. If it is a queue object, the ***isJNDI*** checkbox must be selected to ensure the reference is to a queue object.

- **Message**: The message can be typed into the *Message Field* or can be taken from a file (selected using the file... button). If the data is read from a file, the file path is displayed in the *Message Field.* The file path begins with the special characters ***::file:/.*** The tool uses this to determine that the message should be sent from a populated buffer, as opposed to the contents of the *Message Field* itself.

- **JMSType**: This field in keeping with the JMS Specification, allows the message type to be set. If left blank, the tool detects what type of message is being sent and sets this value (RFH2 Messages are discussed in the RFH2/Usr Tab section below).

- **JMSCorrelationID**: The JMSCorrelationID field allows the user to set the correlation ID for the message. This can be used to link one message with another, for instance, linking a response message with its request message. This field can be left blank should the user not require a correlation ID to be set.

- **JMSPriority**: The JMSPriority field allows the user to set the priority of the message. JMS defines a ten-level priority value, with 0 as the lowest priority, and 9 as the highest. In practice, priorities 0-4 are considered as gradations of normal priority, and priorities 5-9 are considered as gradations of expedited priority. This tool defaults to priority level 4.

- **JMSTimeToLive**: The *JMSTimeToLive Field* allows the user to set the default length of time, in milliseconds from its dispatch time that a message should be retained by the messaging system. This value is set to zero (0) by default, which means the message does not have an expiration time.

- **JMSDeliveryMode**: The *JMSDeliveryMode radio buttons* enable the user to set the message producer's default delivery mode. The tool defaults to Non-Persistent.

On invocation of the send command by clicking the *Send Message* button, a message (message sent or exception) is displayed in the Log Window.

Disconnect Command

The *Disconnect* command closes the active QCF connection. A message on the status of the *Disconnect* command (successful, already disconnected, or exception) is displayed on the log board.

The RFH2/Usr Tab

The RFH2/Usr Tab presented in figure 4, allows the user to create an RFH2 header as well as specify user defined properties. The data on the RFH2/Usr Tab must be provided before the *Send Message* button on the *JMSQueueSender* tab is clicked, in order for the properties to be made part of the message data supplied on the *JMSQueueSender* Tab.

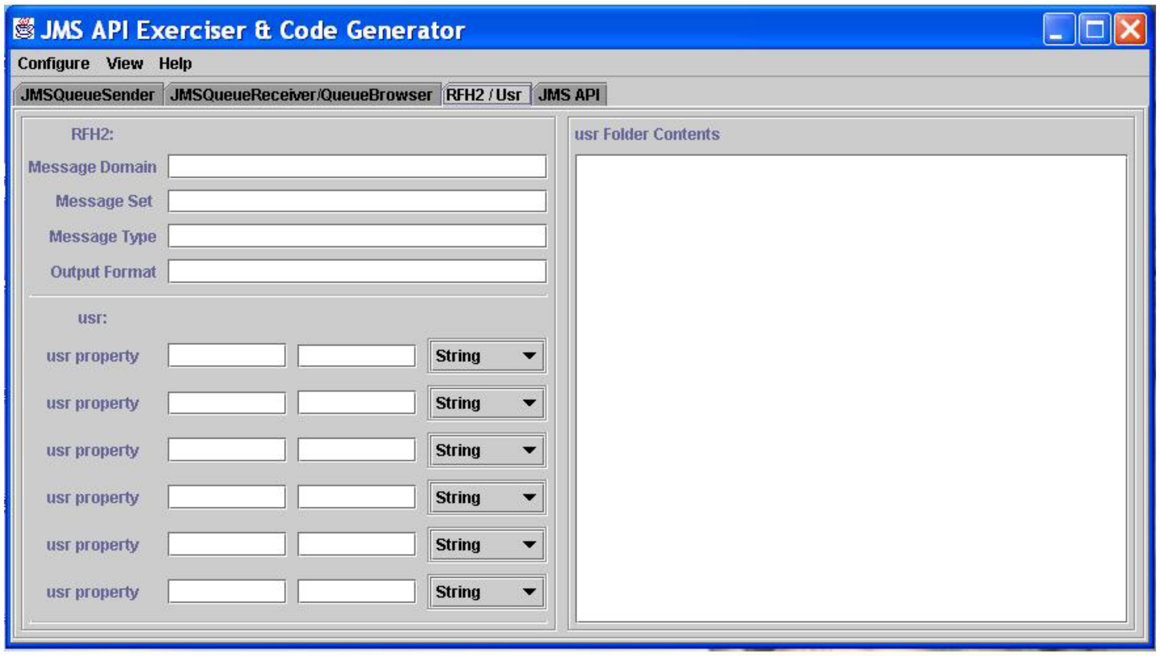


Figure 4: The RFH2/Usr Tab

The RFH2 Window

The RFH2 Window presents the fields where the RFH2 elements are defined. *WebSphere MQ Integrator, Programming Guide version 2.1*, discusses in detail the RFH2 Header and associated elements. The RFH2 fields are:

- Message Domain: The Message Service Domain
- Message Set: The Message Set
- Message Type: The Message Type
- Output Format: The Message Format

It is important to realize that supplying data for the RFH2 header will cause any data entered in the *JMSType* field of the *JMSQueueSender* Tab to be overwritten, with the RFH2 URL.

The Usr Window

The Usr Window enables the user to provide application specific/user defined properties. The tool allows the user to create a maximum of six application/user defined properties. The user (if a user defined property is required) must specify a property name, a property value, and select a property type from the combo box. The supported property types are: Boolean, Double, Float, Integer, and String. The default property type is *String.*

The JMSQueueReceiver/QueueBrowser Tab

The JMSQueueReceiver/QueueBrowser Tab, see Figure 5, provides access to the Message Receiver and Message Browser functionality, enabling the user to receive messages on a queue, or browse the contents of a queue using six commands; Connect, Receive, Browse, Next, EndBrowse, and Disconnect.

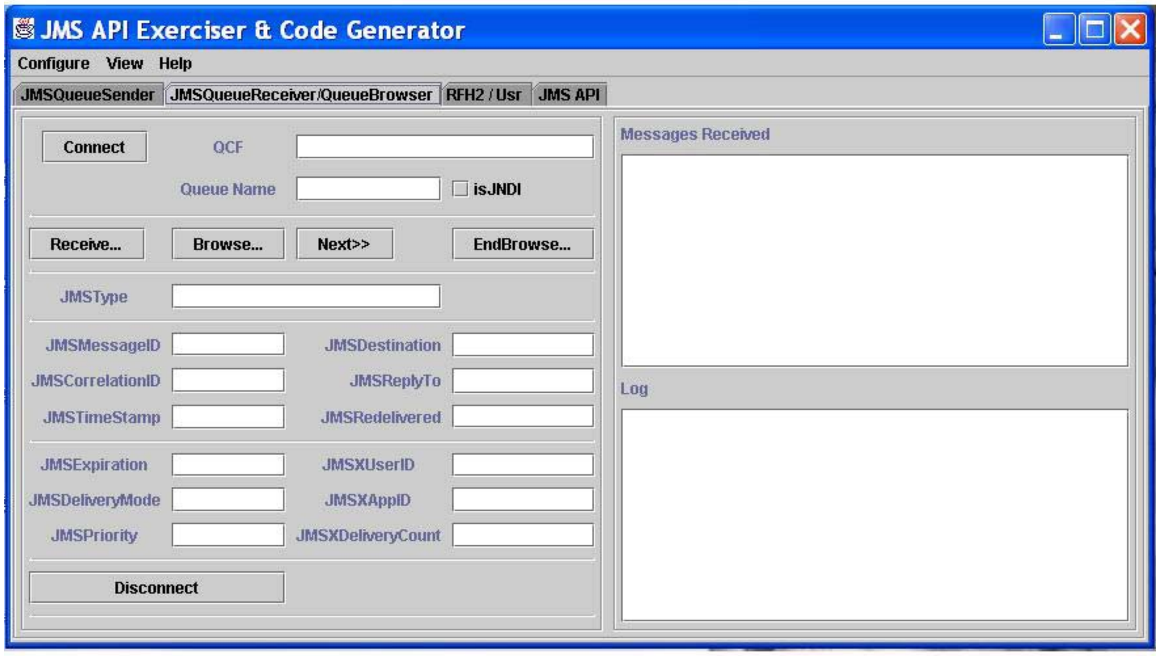


Figure 5: The JMSQueueReceiver/QueueBrowser Tab

Connect Command

The JMSQueueReceiver/QueueBrowser connect command establishes an active connection to the messaging system, and supports the creation of multiple connections. The name of the *QueueConnectionFactory* (QCF) must be supplied to connect. The status of the connection (started or exception) is displayed in the log window. If a connection has already been established using a QCF (possibly on the JMSQueueSender Tab), a message is displayed on the log board to say the connection is already established. As with the JMSQueueSender Tab, ensure that the QCF being used is displayed in the QCF Field before subsequent commands are issued.

Receive Command

The *Receive* command requires that either the name of the Queue object to be retrieved from the JNDI namespace is provided, or that an actual Queue string is supplied. If a queue object is supplied, the ***isJNDI*** checkbox should be selected, otherwise reference to a queue string is assumed. If a queue string is supplied, the queue name must be preceded by a forward slash "/", that is, ***/TestQ*** or ***/TestQMgr/TestQ***.

Once the *Receive* command is issued, a message listener is registered with the specified queue and every message on the queue is received. That is, the message data is displayed in the Messages Received board, and the message header fields are displayed in the different fields on the Tab. Any messages PUT on the queue after the listener has been registered are received and displayed. The JMS API describes the message headers in detail. The JMS Header Fields displayed on the JMSQueueReceiver/QueueBrowser the tool are:

The Message Header Display Fields	
JMSMessageID	JMSCorrelationID
JMSDestination	JMSReplyTo
JMSTimeStamp	JMSRedelivered
JMSExpiration	JMSDeliveryMode
JMSPriority	JMSXUserID
JMSXAppID	JMSXDeliveryCount

Browse Command

As with the receive command, the *Browse* command requires that the Queue object or Queue string is supplied. Once the *Browse… button* is clicked, the tool browses the specified queue for all messages, and stores the messages in an enumeration. The first message (message data and headers) on the queue is displayed. If there is no message on the queue, an exception message is displayed with a suggestion that the queue may be empty.

Next Command

The *Next* command requires that the *Browse* command has already been issued and the first message on the queue has been displayed. The *Next* command displays the next message on a browsed queue. This is achieved by accessing an enumeration of messages, created by the Browser and displaying these messages one at a time, with each click of the *Next* button. If the enumeration created at the time of the browse suggests that there are no more messages on the queue, an exception message is displayed. The *Next* command may not be aware of any messages that may be PUT on the queue after the *Browse* command was issued, as the enumeration is not updated. The *Browse* command has to be issued, in order for new messages that may have been placed on the queue since the last *Browse* call, to be viewed.

EndBrowse Command

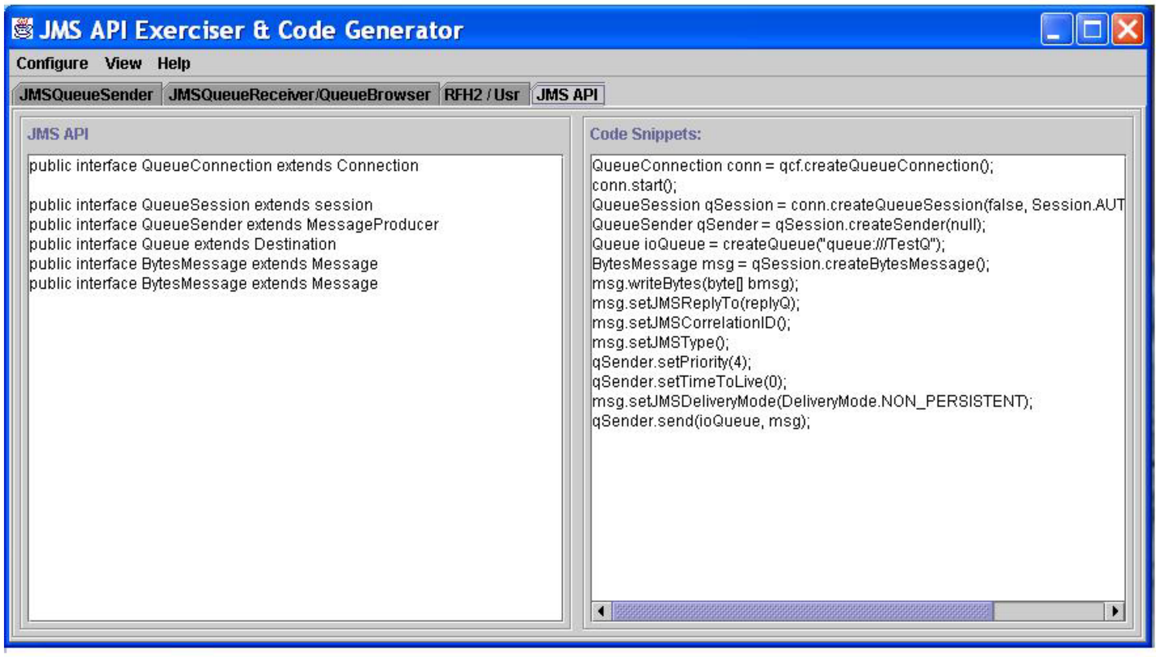
The *EndBrowse* command destroys the enumeration that contains the browsed messages. A message on the status of the *EndBrowse* command (Browser Ended or exception) is displayed on the log board.

Disconnect Command

The *Disconnect* command closes the active QCF connection. A message on the status of the *Disconnect* command (successful, already disconnected, or exception) is displayed on the log board.

The JMS API Tab

The JMS API Tab, see Figure 6, is where all the generated code is displayed. As work is being performed on the other tabs, code is being generated for that work. The code generated shows the JMS calls required but does not involve any Java object manipulation or other non-JMS related program calls that may be required to complete a Java class.



```
public interface QueueConnection extends Connection

public interface QueueSession extends session
public interface QueueSender extends MessageProducer
public interface Queue extends Destination
public interface BytesMessage extends Message
public interface BytesMessage extends Message
```

```
QueueConnection conn = qcf.createQueueConnection();
conn.start();
QueueSession qSession = conn.createQueueSession(false, Session.AUT
QueueSender qSender = qSession.createSender(null);
Queue ioQueue = createQueue("queue:///TestQ");
BytesMessage msg = qSession.createBytesMessage();
msg.writeBytes(byte[] bmsg);
msg.setJMSReplyTo(replyQ);
msg.setJMSCorrelationID();
msg.setJMSType();
qSender.setPriority(4);
qSender.setTimeToLive(0);
msg.setJMSDeliveryMode(DeliveryMode.NON_PERSISTENT);
qSender.send(ioQueue, msg);
```

Figure 6: The JMS API Tab

The JMS API Log Board

The *JMS API Log Board* is where the JMS-defined interfaces and classes in use are presented. The content of this board shows the hierarchy of the classes in use by presenting the classes interface or super classes. The actual generated code that represents the tasks being performed is presented on the Code Snippets Log Board.

The Code Snippets Log Board

The *Code Snippets Log Board* is where the generated code is presented. The hierarchy of each method call can be detected by studying the related entry on the JMS API Log Board. The code is generated with each operation performed using the tool, and as such it is possible to have a non-segregated presentation of the code based on how the user performs their tasks.

Menus

The tools menu options, see Figure 7, are presented:

- **Configure**
  - **Environment** : Environment settings
  - **Close**: Closes the tool
- **View**
  - **Clear**: Clears all Log and Message Boards
  - **Different Look & Feel**
    - **Metal**
    - **CDE / Motif**:
    - **Windows**
- **Help**
  - **About** : About the MA0S SupportPac

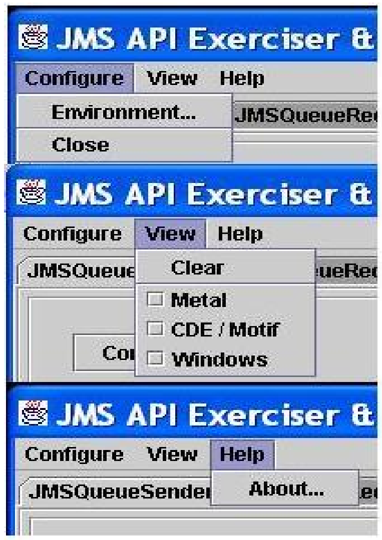


Figure 7: The MA0S Menus.

-- End of Document --