Overview of the JMS Transport Layer

<author: Dave Chappell chappell@sonicsoftware.com>

The JMS transport layer allows SOAP messages to be carried over reliable messaging between Axis clients and Axis services. As illustrated in figure 1, this transport layer can be seamlessly plugged into the Axis engine as an alternative to HTTP. This allows an axis developer to use normal Axis API's to perform synchronous request/reply, document-style message-based request/response, and one-way invocations across a reliable delivery mechanism based on JMS messaging semantics.

Figure 1: SOAP over JMS as an alternate transport to HTTP.

In addition to providing the base plug-in code for the transport, the JMS transport layer also provides a subsystem that allows many conveniences when connecting to a JMS provider, including:

- JMS provider-independent JNDI abstraction support for JMS Administered objects i.e. ConnectionFactories and JMS destinations.
- A mechanism of bypassing JNDI and using direct instantiation of ConnectionFactory and Topic and Queue destinations, for those JMS providers that support such a capability.
- Generic Endpoint abstraction layer for managing Topic and Queue destinations.
- Thread pooling and Session management, providingthread-safe concurrent sending and receiving.

- Automatic reconnect and send-retry based on configurable timeout and retry intervals.
- Request/response management using temporary destinations, and JMSReplyTo semantics.
- An encapsulation of JMS provider-specific information when necessary, such as configuration information, direct instantiation of ConnectionFactories, and JMS provider-specific exception handling details.
- A mechanism for setting JMS specific options as properties in the Call object, and passing them through to the JMS provider during the invocation.
- Pub/Sub and point to point queueing support with durability and persistent send options

As a beginning user of this subsystem, it is not necessary to be intimately aware of the details of all of these capabilities. They are all hidden below the standard Axis Call and Service interfaces, yet there when you need them. Just a few simple setup steps are all you need to get started using the JMS transport layer. Details are provided below.

Setting up the JNDI store for JMS Administered objects

The JMS specification prescribes the use of JNDI for the discovery and instantiation of ConnectionFactory objects, and Topic and Queue destinations. The configuration of the JNDI access and the setup of Topic and Queue destinations is done in a JMS provider-specific fashion. You MUST perform this step in order to proceed to running the example.

In our example, the JNDI access makes use of the file-based FSContext version 1.2 beta 3 release, available from http://java.sun.com/products/jndi/index.html. The minimum required objects to set up are listed in the jndi-connection-factory.properties located in the samples/jms directory. The required fields to set up are:

java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory
java.naming.provider.url=file:///c:/JNDIStore

The JNDI name used to access the ConnectionFactory is “MyCF”, and the lookup name for the sample queue is “MyQ”.

For JMS provider-dependant instructions on how to set this up, see the “JMS Provider Setup Instructions” section of this document.

Running the JMS sample

The JMS sample found in samples/jms, is an invocation of an external stock quote service. For the purpose of keeping the example simple, it is a two-part invocation that first sends the request across the JMS transport to a local JMS listener, which in turn invokes another call to an external service across HTTP (see figure 2).

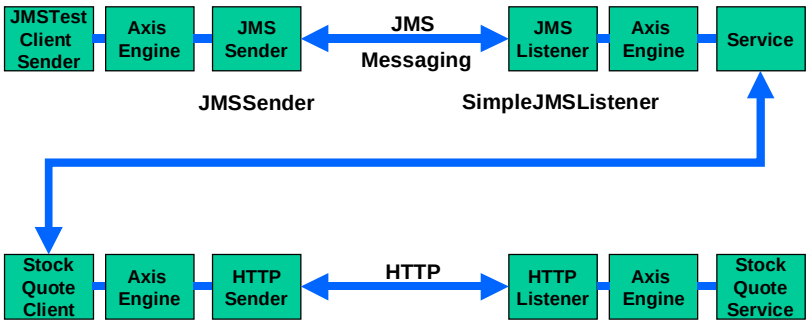

Figure 2: The JMSTest example.

Running the sample with a -? As a command line option will print a usage message showing the different options available. To run the test with the supplied defaults:

From the samples directory, execute the following from the command line:
java samples.jms.JMSTest -c jndi-connection-factory.properties -d "MyQ" IBM PRGS

This will send the SOAP request across the JMS transport layer, to the local instance of SimpleJMSListener. SimpleJMSListener is a JMS Listener that receives the SOAP message, which in turn will invoke the Axis engine to call the StockQuote service. This service will then in turn invoke samples.stock.StockQuoteService. The StockQuoteService will make a HTTP request across the Internet to invoke a web service that is hosted at an external web site.

If you do not have a live Internet connection, then specifying "XXX" as a ticker symbol. This will skip the external invocation and return a hard-coded value. This is sufficient to test the JMS transport aspect of the sample

By default, the conversation between JMSSender and SimpleJMSListener can be executed on a local machine for testing purposes, as long as there is a JMS provider installed somewhere that the sender and listener can connect to. However, there is nothing to prevent them from being separated across machines or geographic locations.

Invoking the JMS Transport Across the Internet

The previous example uses JMS locally, then bridges to HTTP to access a public site across the Internet. It is also possible for JMS to travel across the Internet in a secure fashion (see Figure 3). For another example of how to invoke an Axis SOAP request using JMS for the complete end-to-end round trip, using a publicly available JMS-enabled web service, go to http://XMethods.net/AxisJMS and follow the instructions there.

Figure 3: Invoking a XMethods-hosted JMS-enabled Web service

JMS Provider Setup Instructions

This section is intended to include JMS provider-specific instructions for each JMS provider that supports the Axis JMS transport layer.

SonicMQ/SonicXQ

Make sure the Sonic Explorer classpath is aware of the FSContext jar files. Before launching the Sonic Explorer, you can modify the explorer.bat or explorer.sh and add fscontext.jar and providerutil.jar to the SONICMQ_CLASSPATH setting.

Next launch the Sonic Explorer. On Windows it can be launched from the Start Menu->Programs->Sonic Software->SonicMQ->Explorer menu item. On UNIX or Linux, run explorer.sh.

In the Sonic Explorer main window, select the "JMS Administered Object Store" node from the tree control in the left pane of the window. Select the "JNDI Naming Service"

option in the right pane and type in the directory to store the administered objects in. For example, if the file system directory to be used is c:\JNDIStore, type the following in the "Properties: " field:

java.naming.provider.url=file:///c:/JNDIStore,java.naming.factory.initial=com.sun.jndi.fscontext.RefFSContextFactory

Select the "Connect" button . The screen should look like the following (Figure 4):

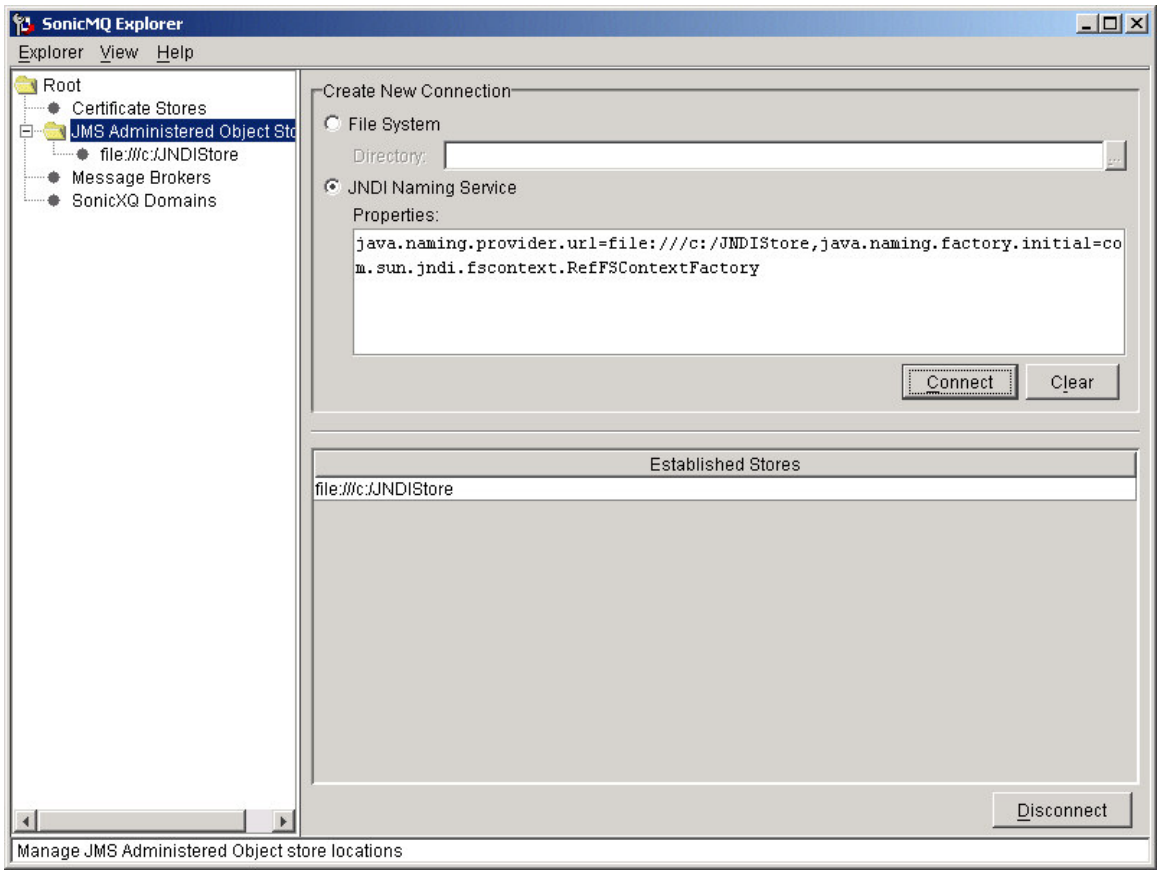

Figure 4: Setting up the SonicMQ Explorer to use Sun's file-based JNDI store.

Select the "file" node in the left pane, and select the "Destinations" tab. Enter the following values as indicated in Figure 5:
The lookup name is "MyQ".
The Type is "Queue".
The Destination Name is "SampleQ1".

Note: SampleQ1 is one of the predefined queues that comes by default in a standard SonicMQ installation. There is no need to further define the queue. The queue name is case sensitive.

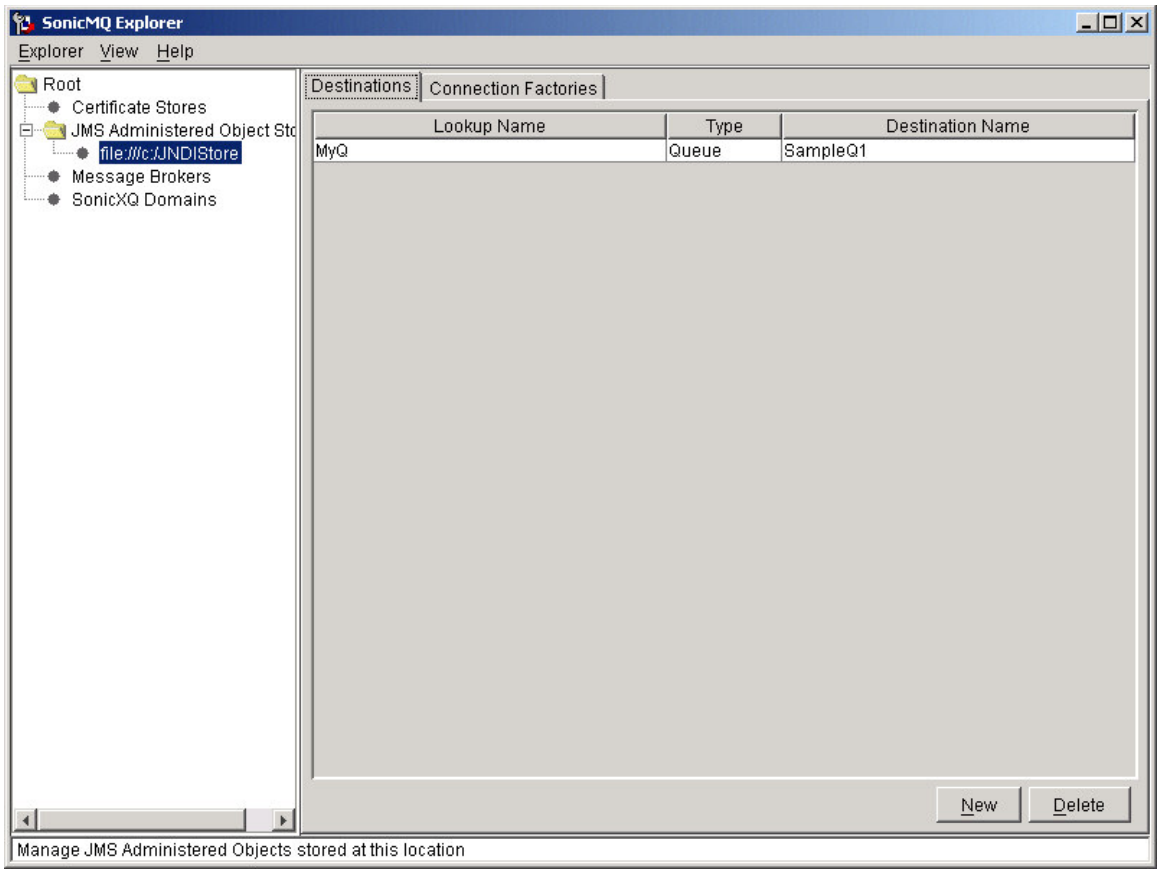

Figure 5: Defining the JNDI lookup name for the "SampleQ1" destination used by the examples.

Select the "Connection Factories" tab. Click on the "New" button, and enter the following information:

Lookup name: MyCF
Factory type: QueueConnectionFactory
URL(s): localhost:2506

All the rest of the fields can be left as is (Figure 6).

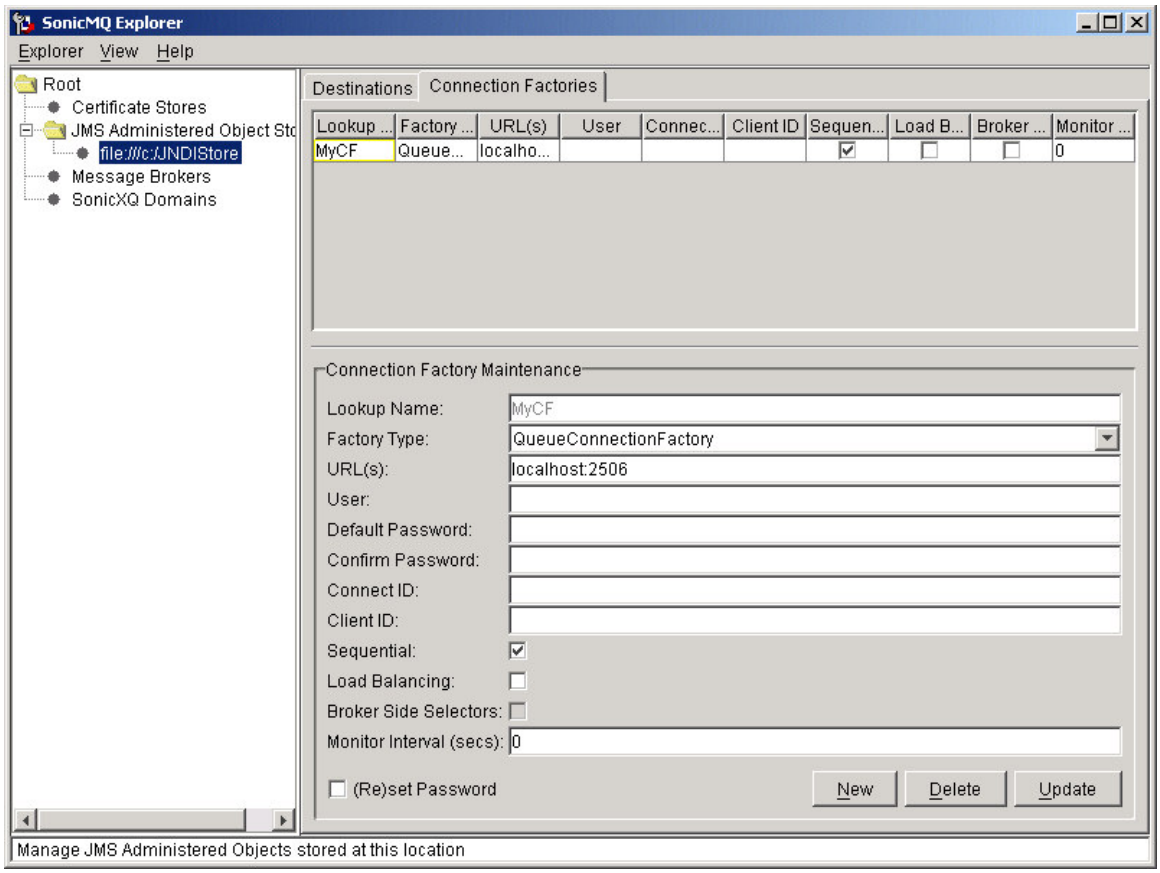

Figure 6: Defining the JNDI lookup name for the ConnectionFactory using the SonicMQ Explorer.