Mercury User Guide

Mercury is an Axis2 module which provides the WS-RM functionality to Axis2. Therefore it is necessary to have a knowledge about Axis2 is required to start work with the Mercury.

Installing Mercury

This can simply be done by adding the mercury-mar-SNAPSHOT.mar to modules folder under the repository and mercury-core-SNAPSHOT.jar to lib folder. At the server side it can be engaged to a service by adding a reference in the services.xml
(eg. <module ref="Mercury"/>). At the client side it can be done by just engaging a module to service client. (eg. serviceClient.engageModule("Mercury");).

Starting a sequence

A sequence can be started by sending a normal message to a Mercury engaged service.

```
try {
        ConfigurationContext configurationContext =
              ConfigurationContextFactory.createConfigurationContextFromFileSystem(
                    AXIS2_REPOSITORY_LOCATION, AXIS2_CLIENT_CONFIG_FILE);
        ServiceClient serviceClient = new ServiceClient(configurationContext, null);
        serviceClient.setTargetEPR(new EndpointReference("http://localhost:
8088/axis2/services/InteropServiceRM"));
        serviceClient.getOptions().setAction("urn:TestAction");

        serviceClient.engageModule("Mercury");
        OMElement omElement = serviceClient.sendReceive(getTestOMElement("Key1" + " " + 1 + "
"));
        System.out.println("OMElement ==> " + omElement);

        MercuryClient mercuryClient = new MercuryClient(serviceClient);
        mercuryClient.terminateSequence("Key1");

        try {
           System.out.println("Waiting thread to sleep");
           Thread.sleep(20000);
        } catch (InterruptedException e) {
```

```
        }
    } catch (AxisFault axisFault) {
        axisFault.printStackTrace();
    }
```

Terminating the sequence

A sequence can be terminated in two different ways.

1. Sending a terminate sequence message

   ```
   MercuryClient mercuryClient = new MercuryClient(serviceClient);
   mercuryClient.terminateSequence("Key1");
   ```

2. Indicating the last message

   ```
   serviceClient.getOptions().setProperty(MercuryClientConstants.LAST_MESSAGE,
   Constants.VALUE_TRUE);
   ```

Sending messages with multiple sequences

Sometimes users may want to send messages with multiple sequences to a same destination. In that case an internal key can be used to indicate the different sequences to Mercury.

eg.

```
serviceClient.getOptions().setProperty(MercuryClientConstants.INTERNAL_KEY,    Key1   );
OMElement omElement = serviceClient.sendReceive(getTestOMElement("Key1" + " " + 1 + " "));
System.out.println("OMElement ==> " + omElement);
MercuryClient mercuryClient = new MercuryClient(serviceClient);
mercuryClient.terminateSequence("Key1");

serviceClient.getOptions().setProperty(MercuryClientConstants.INTERNAL_KEY,    Key2   );
OMElement omElement = serviceClient.sendReceive(getTestOMElement("Key2" + " " + 1 + " "));
System.out.println("OMElement ==> " + omElement);
MercuryClient mercuryClient = new MercuryClient(serviceClient);
mercuryClient.terminateSequence("Key2");
```

Using persistence

Persistence support is only for In only operations. Use the following steps to use persistence with a HSQL data base.

1. Uncomment the rmPersistanceManager parameter in the module.xml file.
   <parameter name="rmPersistanceManager"
   locked="false">org.wso2.mercury.persistence.hibernate.HibernatePersistenceManager</parameter>
2. Add the mercury-persistence-SNAPSHOT.jar to class path (both server and client).
3. Currently Mercury has only one Persistence Manager interface implementation using hibernate. Therefore add all the hibernate jars (use a hibernate distribution) to class path(both server and client).
4. Similarly add all the HSQL jars to class path(both server and client).
5. Start the HSQL Database
6. Send the sequence as in previous case.