How to write a HPC Soam service

This document will demonstrate how to write a HPC Soam service program.

# Prerequisites

1. JDK 1.8.0.60  
   Download Java SE from <http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html> .
2. CXF 2.7.17 (Warning: 3.x versions of CXF do not work.)  
   Download Apache CXF from <http://cxf.apache.org/download.html> .
3. HPC Soam Java source code  
   Download the latest HPC Soam Java source code from <https://github.com/coolmay/HPCSOA/> .
4. Microsoft HPC cluster  
   Please refer to [this](http://go.microsoft.com/fwlink/?LinkId=617894) to deploy a HPC Pack Cluster on Microsoft Azure.

# Interfaces and base classes

This section introduces some interfaces and base classes which will be used to write the HPC Soam service.

You can find all these files at *HPCSOA\sample\HpcSoam\src\com\microsoft\hpc\soam* .

## Class ServiceContainer

java.lang.Object  
 com.microsoft.hpc.soam.ServiceContainer

public abstract class **ServiceContainer** extends java.lang.Object

The application developer must extend this class to create their own service. The developer must implement the desired methods to allow the HPC Soam Host service to interact with the Service Instance:

* onInvoke (mandatory)
* onCreateService (optional)
* onDestroyService (optional)
* onSessionEnter (optional)
* onSessionLeave (optional)
* onApplicationAttach (optional)
* onApplicationDetach (optional)

**Methods**

|  |
| --- |
| ServiceContainer |
| public ServiceContainer() |
| Default Constructor |

|  |
| --- |
| onInvoke |
| public abstract void onInvoke([TaskContext](#_Class_TaskContext) taskContext)  throws [SoamException](#_Class_SoamException) |
| The HPC Soam Host service triggers the invocation of this handler every time a task input is sent to the service to be processed.  The actual service logic should be implemented in this method. This is the only method that is mandatory for the application developer to implement. |
| **Parameters:**  taskContext - Context for the task passed from the client application to the service. |
| **Throws:**  [SoamException](#_Class_SoamException) - The application developer can throw SoamException from this method to indicate that an error occurred during task processing. |
| **See Also:**  [TaskContext](#_Class_TaskContext) |

|  |
| --- |
| onCreateService |
| public void onCreateService([ServiceContext](#_Class_ServiceContext_(not) serviceContext)  throws [SoamException](#_Class_SoamException) |
| The HPC Soam Host service triggers the invocation of this handler just after a Service Instance is started.  The application developer must put any service initialization into this handler. Default implementation does nothing. |
| **Parameters:**  [ServiceContext](#_Class_ServiceContext_(not) - The context contains the information about the service that is required to create a service instance, and all of the functionality that a service will require throughout its lifetime. |
| **Throws:**  [SoamException](#_Class_SoamException) - The application developer can throw SoamException from this method to indicate that an error occurred during the creation/initializion of the service instance. |
| **See Also:**  [ServiceContext](#_Class_ServiceContext_(not) |

|  |
| --- |
| onDestroyService |
| public void onDestroyService()  throws [SoamException](#_Class_SoamException) |
| The HPC Soam Host service triggers the invocation of this handler just before a Service Instance is destroyed.  This handler should do any cleanup for any resources that were used in the onCreateService() method. Exceptions that are thrown in onDestroyService are not returned to the client, they are logged in the SIM log file. Default implementation does nothing. |
| **Throws:**  [SoamException](#_Class_SoamException) - The application developer can throw SoamException from this method to indicate that an error occurred during the destruction/uninitializion of the service instance. |
| **See Also:**  onCreateService(ServiceContext) |

|  |
| --- |
| onSessionEnter |
| public void onSessionEnter([SessionContext](#_Class_SessionContext_(not) sessionContext)  throws [SoamException](#_Class_SoamException) |
| The HPC Soam Host service triggers the invocation of this handler to bind the Service Instance to its owning Session when common data is provided by the Client.  The application developer must put any session-specific initialization into this handler.  If any common data is available for the associated Session, it should be accessed in the developer's implementation of this method. Default implementation of this handler does nothing.  This handler may be called multiple times during the lifecycle of the service instance, since a service instance can be re-assigned to other sessions. |
| **Parameters:**  sessionContext - The context contains information about the binding Session |
| **Throws:**  [SoamException](#_Class_SoamException) - The application developer can throw SoamException from this method to indicate that an error occurred during session-specific initialization or during common data processing. |
| **See Also:**  [SessionContext](#_Class_SessionContext_(not), onSessionLeave() |

|  |
| --- |
| onSessionLeave |
| public void onSessionLeave()  throws [SoamException](#_Class_SoamException) |
| The HPC Soam Host service triggers the invocation of this handler to unbind the Service Instance from its owning Session.  This handler should do any cleanup for any resources that were used in the onSessionEnter() method. Exceptions that are thrown in onSessionLeave are not returned to the client, they are logged in the SIM log file. Default implementation does nothing. |
| **Throws:**  [SoamException](#_Class_SoamException) - The application developer can throw SoamException from this method to indicate that an error occurred while executing the logic to unbind the service instance from its binding session. |
| **See Also:**  onSessionEnter(ServiceContext) |

|  |
| --- |
| onApplicationAttach (not implemented) |
| public void onApplicationAttach([ServiceContext](#_Class_ServiceContext_(not) serviceContext)  throws [SoamException](#_Class_SoamException) |
| The HPC Soam Host service triggers the invocation of this handler when this global standby service is attached to an application.  The application developer must put any service initialization related to attaching to a new application into this handler. Default implementation does nothing. |
| **Parameters:**  serviceContext - The context contains the information about the service that is required to create a service instance, and all of the functionality that a service will require throughout its lifetime. |
| **Throws:**  [SoamException](#_Class_SoamException) - The application developer can throw SoamException from this method to indicate that an error occurred during the creation/initializion of the service instance. |
| **See Also:**  [ServiceContext](#_Class_ServiceContext_(not) |

|  |
| --- |
| onApplicationDetach (not implemented) |
| public void onApplicationDetach()  throws [SoamException](#_Class_SoamException) |
| The HPC Soam Host service triggers the invocation of this handler just before the service is detached from an application and goes into global standby service mode.  This handler should do any cleanup for any resources that were used in the onCreateService() method that are no longer necessary during global standby mode. Default implementation does nothing. |
| **Throws:**  [SoamException](#_Class_SoamException) - The application developer can throw SoamException from this method to indicate that an error occurred during the destruction/uninitializion of the service instance. |

## Class ServiceContext

java.lang.Object  
 com.microsoft.hpc.soam.ServiceContext

public class **ServiceContext** extends java.lang.Object

Provides functionality that the service will require throughout its lifetime. This object will be passed to a service instance when it is created. The service instance may choose to cache a reference to this object and call on it anytime it is required.

**Methods**

|  |
| --- |
| getServiceName |
| public java.lang.String getServiceName()  throws [SoamException](#_Class_SoamException) |
| Gets the HPC Soam service name. |
| **Returns:**  The HPC Soam service name. |
| **Throws:**  [SoamException](#_Class_SoamException) |

|  |
| --- |
| getApplicationName |
| public java.lang.String getApplicationName()  throws [SoamException](#_Class_SoamException) |
| Gets the HPC Soam host service name. |
| **Returns:**  The HPC Soam host service name. |
| **Throws:**  [SoamException](#_Class_SoamException) |

|  |
| --- |
| getDeployDirectory |
| public java.lang.String getDeployDirectory()  throws [SoamException](#_Class_SoamException) |
| Gets the SOA\_HOME directory path. |
| **Returns:**  The SOA\_HOME directory path. |
| **Throws:**  [SoamException](#_Class_SoamException) |

|  |
| --- |
| getLogDirectory |
| public java.lang.String getLogDirectory()  throws [SoamException](#_Class_SoamException) |
| Gets the log directory. |
| **Returns:**  The log directory configured for this application. |
| **Throws:**  [SoamException](#_Class_SoamException) |

## Class SessionContext

java.lang.Object  
 com.microsoft.hpc.soam.SessionContext

public class **SessionContext** extends java.lang.Object

A container for the information associated with a particular Session. Such information may be required during the servicing of a task from a Session.

**Methods**

|  |
| --- |
| getSessionId |
| public java.lang.String getSessionId()  throws [SoamException](#_Class_SoamException) |
| Returns the identifier for the associated Session. |
| **Returns:**  The identifier for the associated Session |
| **Throws:**  [SoamException](#_Class_SoamException) |

|  |
| --- |
| populateCommonData |
| public void populateCommonData(Message commonData)  throws [SoamException](#_Class_SoamException) |
| Populates commonData with the data contained in the Message that was passed in when the Session was created. |
| **Parameters:**  commonData - The message to populate. |
| **Throws:**  [SoamException](#_Class_SoamException) |

|  |
| --- |
| getCommonData |
| public byte[] getCommonData()  throws [SoamException](#_Class_SoamException) |
| Retrieves the data bytes that was passed in when the Session was created. |
| **Returns:**  The common data bytes. |
| **Throws:**  [SoamException](#_Class_SoamException) |

|  |
| --- |
| discardCommonData |
| public void discardCommonData()  throws [SoamException](#_Class_SoamException) |
| Frees the local copy of the common data. |
| **Throws:**  [SoamException](#_Class_SoamException) |

## Class TaskContext

java.lang.Object  
 com.microsoft.hpc.soam.TaskContext

public class **TaskContext** extends java.lang.Object

Provides the context for the task that is bound to the given service invocation.

**Methods**

|  |
| --- |
| populateTaskInput |
| public void populateTaskInput([Message](#_Class_Message) inMsg)  throws [SoamException](#_Class_SoamException) |
| Populates inMsg with the task input that was passed from the client to the service. |
| **Parameters:**  inMsg - The message to populate. |
| **Throws:**  [SoamException](#_Class_SoamException) |

|  |
| --- |
| setTaskOutput |
| public void setTaskOutput([Message](#_Class_Message) outMsg)  throws [SoamException](#_Class_SoamException) |
| Sets the task output message that is to be sent back to the client. |
| **Parameters:**  outMsg - The output message to be returned to the client |
| **Throws:**  [SoamException](#_Class_SoamException) |

## Class Message

java.lang.Object  
 com.microsoft.hpc.soam.Message

public abstract class **Message** extends java.lang.Object

Contains the data that is to be passed by the HPC Soam Host service from the client to the service, or vice versa.

A Message object will use an OutputStream and an InputStream to serialize and deserialize its data, respectively.

The Message class should be extended by the application developer. The developer must implement the onSerialize and onDeserialize methods to ensure proper serialization and deserialization.

**Methods**

|  |
| --- |
| Message |
| public Message() |
| Default Constructor |

|  |
| --- |
| onSerialize |
| public abstract void onSerialize(OutputStream stream)  throws [SoamException](#_Class_SoamException) |
| Invoked when this Message object needs to be serialized. This method is to be implemented by the application developer, such that. Its implementation should serialize the content of this Message to the stream object provided. |
| **Parameters:**  stream - The output stream to be written |
| **Throws:**  [SoamException](#_Class_SoamException) |

|  |
| --- |
| onDeserialize |
| public abstract void onDeserialize(InputStream stream)  throws [SoamException](#_Class_SoamException) |
| Invoked when this Message object needs to be deserialized. This method is to be implemented by the application developer. Its implementation should read the contents of the provided stream and populate this Message object. |
| **Parameters:**  stream - The input stream to be read |
| **Throws:**  [SoamException](#_Class_SoamException) |

## Class SoamException

java.lang.Object  
 java.lang.Throwable  
 java.lang.Exception  
 com.microsoft.hpc.soam.SoamException

public class **SoamException** extends java.lang.Exception

**Methods**

|  |
| --- |
| SoamException |
| public SoamException() |
| Constructs an exception with a null detail message and no cause. The cause may be later initialized by invoking initCause. |

|  |
| --- |
| SoamException |
| public SoamException(java.lang.String message) |
| Constructs an exception with the provided detail message and no cause. The cause may be later initialized by invoking initCause. |
| **Parameters:**  message - The detail message. The detail message can be accessed by invoking getMessage. |

|  |
| --- |
| SoamException |
| public SoamException(java.lang.String message,  int errorCode) |
| Constructs an exception with the provided detail message and associated error code. |
| **Parameters:**  message - The detail message. The detail message can be accessed by invoking getMessage.  errorCode - The Error Code message. The detail message can be accessed by invoking getErrorCode. |

|  |
| --- |
| SoamException |
| public SoamException(java.lang.String message,  java.lang.Throwable cause) |
| Constructs an exception with the provided detail message and cause. |
| **Parameters:**  message - The detail message. The detail message can be accessed by invoking getMessage.  cause - The cause of the exception. The cause can be accessed by invoking getCause. |

|  |
| --- |
| SoamException |
| public SoamException(java.lang.Throwable cause) |
| Constructs an exception with a null detail message and the provided cause. |
| **Parameters:**  cause - The cause of the exception. The cause can be accessed by invoking getCause. |

|  |
| --- |
| getErrorCode |
| public int getErrorCode() |
| Returns the error code associated with the exception. |
| **Returns:**  The error code associated with this exception |

|  |
| --- |
| getErrorType |
| public java.lang.String getErrorType() |
| Returns the type of the exception by name for example "SoamException". |
| **Returns:**  The type of the exception |

# Write the HPC Soam service

This section demonstrates how to write a sample HPC Soam service based on the above introduced interfaces and base classes.

You can find all these sample files at *HPCSOA\sample\SampleService\src\sample\common\* and *HPCSOA\sample\SampleService\src\sample\service\* .

## Message classes

Here we write **MyInput** class which extends [Message](#_Class_Message) to contains the data passed from the client to the service, and **MyOutput** class to contains the data sent from the service to the client.

**MyInput** and **MyOutput** are almost same. They both have some properties and implement the **onSerialize** and **onDeserialize** methods by using JSON to ensure serialization and deserialization.

Below are some code snippets of **MyInput** and **MyOutput**.

Import HPC Soam packages and Jackson packages:

import com.microsoft.hpc.soam.Message;

import com.microsoft.hpc.soam.SoamException;

……

import org.codehaus.jackson.\*;

import org.codehaus.jackson.annotate.JsonProperty;

import org.codehaus.jackson.map.\*;

Extend [Message](#_Class_Message) class:

public class MyInput extends Message {

……

}

Define some properties and marks them as **JsonProperty**:

@JsonProperty("m\_boolean")

private boolean m\_boolean = false;

@JsonProperty("m\_int")

private int m\_int = 123;

@JsonProperty("m\_long")

private long m\_long = 123456L;

@JsonProperty("m\_float")

private float m\_float = 123.456F;

@JsonProperty("m\_double")

private double m\_double = 123.456789;

@JsonProperty("m\_string")

private String m\_string = "This is a sample string from MyInput.";

@JsonProperty("m\_bytes")

private byte[] m\_bytes = {0x11, 0x22, 0x33, 0x44};

@JsonProperty("m\_date")

private Date m\_date = new Date();

Implement **onSerialize**:

@Override

public void onSerialize(OutputStream stream) throws SoamException {

try {

String json = this.toJson();

stream.write(json.getBytes());

} catch (Exception ex) {

throw new SoamException(ex);

}

}

Implement **onDeserialize**:

@Override

public void onDeserialize(InputStream stream) throws SoamException {

try {

ByteArrayOutputStream outStream = new ByteArrayOutputStream();

byte[] data = new byte[4096];

int count = -1;

while ((count = stream.read(data, 0, 4096)) != -1) {

outStream.write(data, 0, count);

}

data = null;

String json = new String(outStream.toByteArray(), "ISO-8859-1");

MyInput my = MyInput.fromJson(json);

this.m\_boolean = my.isBoolean();

this.m\_int = my.getInt();

this.m\_long = my.getLong();

this.m\_float = my.getFloat();

this.m\_double = my.getDouble();

this.m\_string = my.getString();

this.m\_bytes = my.getBytes();

this.m\_date = my.getDate();

} catch (Exception ex) {

throw new SoamException(ex);

}

}

Add Json Parser:

private static final ObjectMapper mapper = new ObjectMapper();

public String toJson() throws JsonGenerationException, JsonMappingException, IOException {

return mapper.writeValueAsString(this);

}

public static MyInput fromJson(String json) throws JsonParseException, JsonMappingException, IOException {

if (json == null) {

return null;

} else {

return mapper.readValue(json, MyInput.class);

}

}

## Service classes

Here we write **MyService** class which extends [ServiceContainer](#_Class_ServiceContainer) to create our own service. we implement the **onInvoke** methods to process the **MyInput** data passed from client and send a **MyOutput** data back.

Below are some code snippets of **MyService**.

Import HPC Soam packages:

import com.microsoft.hpc.soam.\*;

Extend [ServiceContainer](#_Class_ServiceContainer) class:

public class MyService extends ServiceContainer {

……

}

Implement **onInvoke**:

In this method, we do the following things:

* Try to get the common data from [SessionContext](#_Class_SessionContext).
* Call [TaskContext](#_Class_TaskContext).**populateTaskInput()** to get the **MyInput** data, whichinvokes **MyInput**.**onDeserialize()** to deserialize the binary data passed from client and generate the **MyInput** data.
* Create a **MyOutput** object and set something.
* Call [TaskContext](#_Class_TaskContext).**setTaskOutput()** to set the **MyOutput** data, whichinvokes **MyOutput**.**onSerialize()** to serialize the **MyOutput** object to binary data which will be sent back to client.

@Override

public void onInvoke(TaskContext taskContext) throws SoamException {

/\*\*

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Do your service logic here. This call applies to each task

\* submission.

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

// Get the input that was sent from the client

try {

int factor = 10;

byte[] commonData = m\_sessionContext.getCommonData();

if (commonData != null) {

factor = commonData.length;

}

MyInput myInput = new MyInput();

taskContext.populateTaskInput(myInput);

// Set the output

MyOutput myOutput = new MyOutput();

myOutput.setBoolean(!myInput.isBoolean());

myOutput.setInt(myInput.getInt() \* factor);

myOutput.setFloat(myInput.getFloat() \* factor);

myOutput.setDouble(myInput.getDouble() \* factor);

myOutput.setLong(myInput.getLong() \* factor);

myOutput.setString(m\_serviceContext.getServiceName() + " returned: "

+ myInput.getString().toUpperCase());

myOutput.setBytes(commonData);

myOutput.setDate(new Date());

// Set our output message

taskContext.setTaskOutput(myOutput);

} catch (Exception ex) {

throw new SoamException(ex);

}

}

# Client classes

This section introduces some classes which will be used to write the HPC Soam client.

You can find all these files at *HPCSOA\HpcServiceHost\src\com\microsoft\hpc\scheduler\session*.

## Class SessionStartInfo

java.lang.Object  
 com.microsoft.hpc.scheduler.session.SessionStartInfo

public class **SessionStartInfo** extends java.lang.Object

Provides the information for creating a session.

**Methods**

|  |
| --- |
| SessionStartInfo |
| public SessionStartInfo(String headnode, String service,  String username, String password) |
| Constructor. |
| **Parameters:**  headnode - The host name of the HPC Head node to connect.  service - The service name of the SOA service.  username - The user name to run HPC job.  password - The password for the specified user. |

|  |
| --- |
| setSecure |
| public void setSecure(Boolean value) |
| Sets whether to use secure transport between client and HPC Broker nodes. |
| **Parameters:**  value - TRUE to use secure transport and FALSE not to use. |

|  |
| --- |
| setCommonDataClientId |
| public void setCommonDataClientId(String dataClientId) |
| Sets the common data client Id to session start info. The common data client Id is a unique Id string which identifies the DataClient. The HPC Soam service retrieves the common data from HPC data server with this id. |
| **Parameters:**  dataClientId - The unique common data client Id. |

## Class Session

java.lang.Object  
 com.microsoft.hpc.scheduler.session.SessionBase  
 com.microsoft.hpc.scheduler.session.Session

public class **Session** extends com.microsoft.hpc.scheduler.session.SessionBase

A service session.

**Methods**

|  |
| --- |
| createSession |
| public static Session createSession(SessionStartInfo startInfo)  throws SocketTimeoutException, SessionException |
| Create a service session object. |
| **Parameters:**  startInfo - The session start info for creating the service session. |
| **Returns:**  A service job session object. |
| **Throws:**  SocketTimeoutException  SessionException |

|  |
| --- |
| close |
| public void close()  throws SocketTimeoutException, SessionException |
| Close the session. |
| **Throws:**  SocketTimeoutException  SessionException |

## Class DurableSession

java.lang.Object  
 com.microsoft.hpc.scheduler.session.SessionBase  
 com.microsoft.hpc.scheduler.session.DurableSession

public class **DurableSession** extends com.microsoft.hpc.scheduler.session.SessionBase

A durable session whose requests and responses gets stored in MSMQ.

**Methods**

|  |
| --- |
| createSession |
| public static DurableSession createSession(SessionStartInfo startInfo)  throws SocketTimeoutException, SessionException |
| Create a service durable session object. |
| **Parameters:**  startInfo - The session start info for creating the service session. |
| **Returns:**  A durable session object. |
| **Throws:**  SocketTimeoutException  SessionException |

|  |
| --- |
| close |
| public void close()  throws SocketTimeoutException, SessionException |
| Close the session. |
| **Throws:**  SocketTimeoutException  SessionException |

## Class DataClient

java.lang.Object  
 com.microsoft.hpc.scheduler.session.DataClient

public class DataClient extends java.lang.Object

Provides interfaces to manage(create/open/delete) and access(write/read) DataClient on the data server.

**Methods**

|  |
| --- |
| create |
| public static DataClient create(String dataclientid, String headnode,  String username, String password)  throws DataException |
| Create a new instance of DataClient. |
| **Parameters:**  dataclientid - The string identifier of the DataClient.  headnode - The host name of the HPC Head node to connect.  username - The user name to run HPC job.  password - The password for the specified user. |
| **Returns:**  A new DataClient instance. |
| **Throws:**  DataException |

|  |
| --- |
| open |
| public static DataClient open(String dataclientid, String headnode,  String username, String password)  throws DataException |
| Open an existing DataClient with the specified ID. |
| **Parameters:**  dataclientid - The string identifier of the DataClient.  headnode - The host name of the HPC Head node to connect.  username - The user name to run HPC job.  password - The password for the specified user. |
| **Returns:**  A DataClient instance that provides read/write access to the specified DataClient. |
| **Throws:**  DataException |

|  |
| --- |
| open |
| public static DataClient open(String dataserveraddress, String dataclientid)  throws DataException |
| Open a data client on specified data server (file share). |
| **Parameters:**  dataserveraddress - File share root path.  dataclientid - The string identifier of the DataClient. |
| **Returns:**  A DataClient instance that provides read/write access to the specified DataClient. |
| **Throws:**  DataException |

|  |
| --- |
| delete |
| public static void delete(String dataclientid, String headnode,  String username, String password)  throws DataException |
| Delete an existing DataClient with the specified ID. |
| **Parameters:**  dataclientid - The string identifier of the DataClient.  headnode - The host name of the HPC Head node to connect.  username - The user name to run HPC job.  password - The password for the specified user. |
| **Throws:**  DataException |

|  |
| --- |
| close |
| public void close() |
| Close the DataClient. |

|  |
| --- |
| readRawBytesAll |
| public byte[] readRawBytesAll()  throws DataException |
| Read back all the data in the DataClient as raw bytes. |
| **Returns:**  Data in the DataClient as raw bytes. |
| **Throws:**  DataException |

|  |
| --- |
| writeRawBytesAll |
| public void writeRawBytesAll(byte[] data)  throws IllegalStateException, DataException |
| Write raw bytes to the DataClient and mark the DataClient as ready for read. |
| **Parameters:**  data - The bytes to write. |
| **Throws:**  IllegalStateException - If the DataClient is opened for read.  DataException - If there is any I/O error when writing the data. |

|  |
| --- |
| writeRawBytesAll |
| public void writeRawBytesAll(byte[] data, boolean compressible)  throws IllegalStateException, DataException |
| Write raw bytes to the DataClient and mark the DataClient as ready for read. |
| **Parameters:**  data - The bytes to write.  compressible - A flag indicating whether the data is compressible. |
| **Throws:**  IllegalStateException - If the DataClient is opened for read.  DataException - If there is any I/O error when writing the data. |

## Class BrokerClient

java.lang.Object  
 com.microsoft.hpc.scheduler.session.BrokerClient

public class BrokerClient<TContract extends Service> extends java.lang.Object

Used to communicate with the broker to send and receive messages.

**Methods**

|  |
| --- |
| BrokerClient |
| public BrokerClient(SessionBase session, Class<TContract> service) |
| Constructor. |
| **Parameters:**  session - Session or DurableSession.  service - Service class to instantiate. |

|  |
| --- |
| close |
| public void close()  throws SocketTimeoutException, SessionException |
| Closes broker connections. |
| **Throws:**  SocketTimeoutException  SessionException |

|  |
| --- |
| sendRequest |
| public void sendRequest(Object request, Object userData)  throws SessionException, SocketTimeoutException |
| Send typed messages with user data to broker. |
| **Parameters:**  request - Typed message to send.  userData - User supplied request data. (will be converted to String because of Java limitations) |
| **Throws:**  SessionException  SocketTimeoutException |

|  |
| --- |
| endRequests |
| public void endRequests()  throws SessionException, SocketTimeoutException |
| Commits uncommited requests to broker's store, and indicates that all requests are sent out. Default timeout is 60 secends. |
| **Throws:**  SessionException  SocketTimeoutException |

|  |
| --- |
| getResponses |
| public <TMessage> Iterable<BrokerResponse<TMessage>> getResponses(  Class<TMessage> message)  throws SessionException |
| Get the responses from the server. The function returns an iterable instance which can be used to process for each. |
| **Parameters:**  message - Response's type. |
| **Returns:**  An iterable instance which stores all responses. |
| **Throws:**  SessionException |

## Class BrokerResponse

java.lang.Object  
 com.microsoft.hpc.scheduler.session.BrokerResponse

public class BrokerResponse<TMessage> extends java.lang.Object

Wraps response messages to provide access to data, faults, and user data.

**Methods**

|  |
| --- |
| getResult |
| public TMessage getResult()  throws SOAPFaultException, SOAPException |
| Gets response's data. |
| **Returns:**  Response's data. |
| **Throws:**  SOAPFaultException  SOAPException |

|  |
| --- |
| getUserData |
| public String getUserData() |
| Returns user data supplied when sending the response's corresponding request. |
| **Returns:**  Returns data supplied with response's request. |

# Write the HPC Soam client

This section demonstrates how to write a sample HPC Soam client based on the above classes.

You can find the sample file at *HPCSOA\sample\SampleClient\src\HelloWorld.java* .

## Client sample

We write a client function to test the HPC Soam service we wrote above. This function sends several **MyInput** data to the service and waits to get **MyOutput** response data from the service.

Below are some code snippets of the function.

private static int RunSoamTest() {

int nerrs = 0;

int nresponses = 0;

Create a **SessionStartInfo** to specify the Head node name, login username, password and service name. The service name is the base name of the ServiceRegistration file which defines the configuration settings of the service.

SessionStartInfo info = new SessionStartInfo(headnode, serviceName, username, password);

System.out.printf("Creating a session for %s...\n", serviceName);

Create a 1KB size random binary data as Common Data and send to HPC Data Server by invoking **DataClient**.

try {

// Prepare 1k binary data

byte[] data = new byte[1024];

Random r = new Random();

r.nextBytes(data);

// Create common data client

String dataClientId = java.util.UUID.randomUUID().toString();

DataClient dataClient = DataClient.create(dataClientId, headnode, username, password);

System.out.printf("new common data client id = %s\n", dataClientId);

// Write data to Windows HPC Cluster

dataClient.writeRawBytesAll(data, true);

Set the data client id information to **SessionStartInfo.** So the service could download the Common Data for sessions.

// pass data client id to session

info.setCommonDataClientId(dataClientId);

Create a **DurableSession** with the **SessionStartInfo**.

DurableSession session = DurableSession.createSession(info);

System.out.printf("new session id = %d\n", session.getId());

Create a **BrokerClient** with the **DurableSession**. And here **SoamSvc** is the class of HPC Soam Host service.

BrokerClient<SoamSvc> client = new BrokerClient<SoamSvc>(session, SoamSvc.class);

System.out.printf("Sending %d requests...\n", nrequests);

Create **MyInput** data and send as a request.

for (int i = 0; i < nrequests; i++) {

MyInput input = new MyInput();

ObjectFactory of = new ObjectFactory();

SoamInvoke request = of.createSoamInvoke();

request.setSoamInputObject(input);

client.sendRequest(request, i + 1);

System.out.printf("Sent request %s: %s%n", i + 1, input);

}

Finish sending requests.

System.out.println("call endRequests() ...");

client.endRequests();

Call **client.getResponses()** to get responses from the service.

System.out.println("Retrieving responses...");

for (BrokerResponse<SoamInvokeResponse> response : client.<SoamInvokeResponse>getResponses(SoamInvokeResponse.class)) {

nresponses++;

Get data from each response and print to screen.

try {

MyOutput reply = new MyOutput();

response.getResult().getSoamOutputObject(reply);

System.out.printf("\tReceived response for request %s: %s%n", response.getUserData(), reply);

} catch (Exception ex) {

nerrs++;

System.out.printf("Error: process %s-th reuqest: %s%n", response.getUserData(), ex.toString());

}

}

System.out.printf("Done retrieving %d responses%n", nresponses);

Close the client and session.

client.close();

session.close();

} catch (Throwable e) {

nerrs++;

e.printStackTrace();

}

return nerrs;

}

## High throughput sample

In this sample, we demonstrate how to send requests to service in high throughput way.

Below are some code snippets of the function.

private static int RunHighThroughputTest() {

int nerrs = 0;

Create a **SessionStartInfo** to specify the Head node name, login username, password and service name. And set Secure to FALSE to use HTTP mode for transferring between client and HPC Broker nodes.

SessionStartInfo info = new SessionStartInfo(headnode, serviceName, username, password);

info.setSecure(Boolean.FALSE);

System.out.printf("Creating a session for %s...\n", serviceName);

We will use multiple BrokerClients to send data. So we store them in a list.

List<BrokerClient<SoamSvc>> clientList = new ArrayList<BrokerClient<SoamSvc>>();

Create a **Session** with the **SessionStartInfo**.

try {

Session session = Session.createSession(info);

System.out.printf("new session id = %d\n", session.getId());

Invoke **sendRequests**() to create BrokerClients and send data.

sendRequests(session, clientList);

Invoke **getResponses**() to get responses from service.

getResponses(clientList);

Close the session and exit.

session.close();

} catch (Throwable e) {

nerrs++;

e.printStackTrace();

}

return nerrs;

}

In **sendRequests**(), we start BrokerClients in multi-thread.

private static void sendRequests(final SessionBase session, final List<BrokerClient<SoamSvc>> clientList) {

Runnable runnable = new Runnable() {

@Override

public void run() {

try {

String clientId = UUID.randomUUID().toString();

final BrokerClient<SoamSvc> client = new BrokerClient<SoamSvc>(

clientId, session, SoamSvc.class);

clientList.add(client);

// Send request

for (int i = 0; i < nbatchrequests; i++) {

MyInput input = new MyInput();

ObjectFactory of = new ObjectFactory();

SoamInvoke request = of.createSoamInvoke();

request.setSoamInputObject(input);

client.sendRequest(request, i + 1);

//System.out.printf("Sent request %s: %s%n", i + 1, input);

}

client.endRequests(timeout);

} catch (Throwable e) {

e.printStackTrace();

}

}

};

Thread[] threads = new Thread[batchCount];

for (int i = 0; i < batchCount; i++) {

threads[i] = new Thread(runnable);

threads[i].start();

}

for (int i = 0; i < batchCount; i++) {

try {

threads[i].join();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

In **getResponses**(), we get responses in multi-thread.

private static void getResponses(final List<BrokerClient<SoamSvc>> clientList) {

ThreadPoolExecutor threadPool = new ThreadPoolExecutor(batchCount \* nbatchthreads, batchCount \* nbatchthreads, 60, TimeUnit.SECONDS, queue);

Thread[] threads = new Thread[batchCount];

for (int i = 0; i < batchCount; i++) {

final BrokerClient<SoamSvc> client = clientList.get(i);

threads[i] = new Thread(new Runnable() {

@Override

public void run() {

try {

for (final BrokerResponse<SoamInvokeResponse> response : client.<SoamInvokeResponse>getResponses(SoamInvokeResponse.class)) {

// getResult is also expensive, better parallel to improve performance

threadPool.execute(new Runnable() {

@Override

public void run() {

try {

MyOutput reply = new MyOutput();

response.getResult().getSoamOutputObject(reply);

//System.out.printf("\tReceived response for request %s: %s%n", response.getUserData(), reply);

} catch (Exception ex) {

System.out.printf("Error: process %s-th reuqest: %s%n", response.getUserData(), ex.toString());

}

}

});

}

} catch (Throwable e) {

e.printStackTrace();

}

}

});

threads[i].start();

}

for (int i = 0; i < batchCount; i++) {

try {

threads[i].join();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

while (threadPool.getActiveCount() > 0) {

//System.out.printf("Active: %d Completed: %d\n", threadPool.getActiveCount(), threadPool.getCompletedTaskCount());

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

threadPool.shutdown();

System.out.printf("close client...\n");

for (int i = 0; i < batchCount; i++) {

BrokerClient<SoamSvc> client = clientList.get(i);

try {

client.close(true);

} catch (Throwable e) {

e.printStackTrace();

}

}

}

# ServiceRegistration file

This section introduces how to configure the HPC Soam service in the HPC ServiceRegistration file.

You can find the sample file at *HPCSOA\SoamSvcHost\SoamSvc.config* for running on Windows or *HPCSOA\SoamSvcHostLinux\SoamSvcLinux.config* for on Linux.

The configuration of the HPC Soam service is defined in tab **<microsoft.Hpc.Session.ServiceRegistration>** . Below are some snippets of the configuration file.

<microsoft.Hpc.Session.ServiceRegistration>

The attribute **assembly** in the **<service>** tab indicates the path of the jar file of the HPC Soam Host service.

<service assembly="D:\HPCSOA\SoamSvcHost\HpcSoamSvc.jar" includeExceptionDetailInFaults="true"

maxConcurrentCalls="0" serviceInitializationTimeout="60000"

enableMessageLevelPreemption="true" stdError="" maxMessageSize="65536"

soaDiagTraceLevel="Verbose, ActivityTracing">

Environment variables could be defined in **<environmentVariables>** section.

* **CCP\_TASK\_NOTIFY** indicates the job could be canceled by HPC scheduler. So it should always be defined.
* **JAVA\_HOME** indicates the Java home path on the compute nodes.
* **CXF\_HOME** indicates the Apache CXF home path on the compute nodes.
* **SOA\_HOME** indicates the working directory on the compute nodes. The HPC Soam Host service will load configuration and setting files from this directory.
* **ENABLE\_BACKEND\_SECURITY** indicates whether to encrypt the transfer messages between HPC broker nodes and compute nodes. If you set this variable as “true”, you need to do more configuration steps to run the program. See more details in the README file under the root directory on [GitHub](https://github.com/coolmay/HPCSOA/).
* **HPC\_SOAM\_SERVICE\_ASSEMBLY** indicates the path of the jar file of this HPC Soam sample service.
* **HPC\_SOAM\_SERVICE\_NAME** indicates the full name of this HPC Soam sample service class.

<environmentVariables>

<add name="CCP\_TASK\_NOTIFY" value="CTRL\_C" />

<add name="JAVA\_HOME" value="C:\java\jdk1.8.0\_60" />

<add name="CXF\_HOME" value="C:\java\apache-cxf-2.7.17" />

<add name="SOA\_HOME" value="D:\HPCSOA\SoamSvcHost" />

<add name="ENABLE\_BACKEND\_SECURITY" value="false" />

<add name="HPC\_SOAM\_SERVICE\_ASSEMBLY" value="D:\HPCSOA\SoamSvcHost\SampleService.jar" />

<add name="HPC\_SOAM\_SERVICE\_NAME" value="sample.service.MyService" />

</environmentVariables>

</service>

In the **<host>** tab, we set **hostType** as “Customize” and **exeFileName** as the command line to start up the HPC host service.

<host hostType="Customize" exeFileName="java -jar &quot;D:\HPCSOA\SoamSvcHost\Microsoft-HpcServiceHost-3.0.jar&quot; &gt; D:\HPCSOA\SoamSvcHost\win\_${CCP\_TASKINSTANCEID}.log 2&gt;&amp;1" />

</microsoft.Hpc.Session.ServiceRegistration>

# Build and run

In this section, we will build our Java source and run the program on the HPC cluster.

## How to build

On the build machine: (the build machine could be one of your compute nodes)

1. Download and install all "Prerequisites".
2. Switch to HpcServiceHost\src\ directory and edit makejar.cmd file.
3. Make sure JAVA\_HOME, CXF\_HOME, SOA\_HOME are pointed to the correct location.

E.g.  
export JAVA\_HOME=C:\java\jdk1.8.0\_60  
export CXF\_HOME= C:\java\apache-cxf-2.7.17  
export SOA\_HOME=D:\HPCSOA\SoamSvcHost

1. Run "makejar.cmd" without any argument.
2. Do the same things in sample\HpcSoam\src\ , sample\HpcSoamSvc\src\ , sample\SampleService\src\ directories.
3. JAR files should be copied to SOA\_HOME directory automatically.

You should see the following files added into your SOA\_HOME directory.

* Microsoft-HpcServiceHost-3.0.jar
* Microsoft-HpcSession-3.0.jar
* HpcSoam.jar
* HpcSoamSvc.jar
* SampleService.jar

## How to deploy

On all compute nodes:

1. Download and install all "Prerequisites".
2. Copy the SOA\_HOME directory from the build machine.
3. Make sure the JAVA\_HOME, CXF\_HOME, SOA\_HOME are as same as on the build machine.

On head node and broker nodes:

1. Copy the ServiceRegistration file in the SOA\_HOME directory to %CCP\_HOME%\ServiceRegistration directory. HPC will load this file there.

(The default path of %CCP\_HOME% should be C:\Program Files\Microsoft HPC Pack 2012\)

## How to run

On client machine: (the client machine could be one of your compute nodes)

1. Go to sample/SampleClient/src directory.
2. Edit RunTest.cmd to set JAVA\_HOME, CXF\_HOME and SOA\_HOME to correct path.
3. Edit HelloWorld.java to set correct username, password, headnode name, and service name. the service name should be the base name of the ServiceRegistration file.
4. Run "RunTest.cmd src" to build the HelloWorld.java.
5. Run "RunTest.cmd run" to run the client.

End of Document