WebServices with OC4J

Simple Development with Oracle's J2EE Environment

Developed by: transentia

Unit 1, 40 Skew St.
Sherwood
Queensland 4075
Australia

Mobile: 0416 078 813
Phone/FAX: (07) 3278 5890
Email: contact@transentia.com.au
WWW: www.transentia.com.au

ACN: 091-142-880
ABN: 680-911-428-80

Exercise

5

WebServices with OC4J

Simple Development with Oracle's J2EE Environment

Introduction

This session shows how to develop Oracle WebServices "by hand." Effectively, you will be going "behind-the-scenes" and looking at the basic functionality of Oracle's OC4J system. Tools like JDeveloper cover up much of the requisite activity, but it is often useful to see what is going on.

In this session you will use Oracle's OC4J J2EE container "by hand" to create and execute a simple WebService. You will also develop a simple client that makes use of the WebService.

Setting Up

In this session you will need:

- Oracle's OC4J 9.0.3

  OC4J is incorporated into JDeveloper 9i and so the easiest way to get this will be to install JDeveloper. JDeveloper 9i 9.0.3 (or greater) will be provided to you and should be installed prior to starting the exercise below.

  Installation is easy: Simply unzip the supplied file to create a directory hierarchy rooted at **C:\jdev903**.

Populate Development Directories

Open a Command Prompt window and then execute the following command to set up a new directory for you to 'play' in (in this sheet the DOS command prompt is shown as **>**, *you should not type this directly*. This command assumes that your CD-ROM is accessible via drive Z:):

```
> xcopy /e/i "Z:\Exercises\5 OC4JWS\framework" C:\OC4JWS
```

This directory contains a number of empty files for you to edit, alongside a number of 'boilerplate' files that you can examine but that should not be changed.

You should do all your work within the **C:\OC4JWS** directory.

Simple WebService

Much of the "business functionality" of the application you will build is contained in a very simple Java class.

Create the Bean Class

Edit the empty skeleton file **C:\OC4JWS\Haikus\src\transentia\HaikuWS.java**. The content should be as follows:

```
package transentia;

public class HaikuWS
  {
  private Haiku[] haikus =
    {
    new Haiku("I'm sorry, there's -- um --\n" +
              "  insufficient -- what's-it-called?\n" +
              "    The term eludes me ...",
              "Owen Mathews"),
    new Haiku ("The code was willing,\n" +
               "  It considered your request,\n" +
               "    But the chips were weak.",
               "Barry L. Brumitt"),
    new Haiku("The Tao that is seen\n" +
              "  Is not the true Tao, until\n" +
              "    You bring fresh toner.",
              "Bill Torcaso")
    };

  // This method is exposed as a web service.
  public Haiku getHaiku()
    {
    return (haikus[(int)(Math.rint(Math.random() *
                  (haikus.length - 1)))]);
    }
}
```

Points to Note

- this is a very simple Java class…it contains no infrastructurally imposed overhead code

Create the Interface Definition File

The interface defines the functionality that is to be exposed as a web service.

Edit the empty skeleton file **C:\OC4JWS\Haikus\src\transentia\IHaikuWS.java**. The content should be as follows:

```
package transentia;

public interface IHaikuWS
  {
  public Haiku getHaiku();
  }
```

Create the Haiku JavaBean

The Haiku JavaBean is a simple package for the data that constitutes a Haiku: a string containing the poem itself and a string containing the author's name.

Edit the supplied skeleton file **C:\OC4JWS\Haikus\src\transentia\Haiku.java**. The content should be as follows:

```
package transentia;

import java.util.StringTokenizer;
import java.io.Serializable;

public class Haiku
  implements Serializable
  {
  public String haiku;
  public String author;

  public Haiku()
    {
    this("","");
    }

  public Haiku(String newHaiku, String newAuthor)
    {
    haiku = newHaiku;
    author = newAuthor;
    }

  public String getHaiku()
    {
    return (haiku);
    }

  public void setHaiku (String haiku)
    {
    this.haiku = haiku;
    }

  public String getAuthor()
    {
    return (author);
    }

  public void setAuthor (String author)
    {
    this.author = author;
    }

  public String toString()
    {
    return (haiku + "\n" + author);
    }
  }
```

Create the WebServices Generator Tool Configuration File

Like most vendors, Oracle supplies a simple tool that is used to generate automatically all the various pieces of infrastructure required by a WebService. Oracle's "WebServicesAssembler" tool is configured by a simple XML file.

Edit the config file **C:\OC4JWS\Haikus\config.xml** to become as follows:

```
<web-service>
  <display-name>Haiku Web Service</display-name>
  <description>Haiku WebServices Exercise</description>

  <destination-path>./haikuws.ear</destination-path>

  <temporary-directory>./tmp</temporary-directory>

  <context>/haikuws</context>

  <stateless-java-service>
    <interface-name>transentia.IHaikuWS</interface-name>
    <class-name>transentia.HaikuWS</class-name>
    <uri>HaikuWS</uri>
    <java-resource>./classes</java-resource>
  </stateless-java-service>
</web-service>
```

Points to Note

- The WebServicesAssembler tool will automatically create an "Enterprise ARchive" file whose name is specified in the config.xml file (./haikuws.ear).

Build and Execute I

Time to go!

You have been provided with a number of build/execute scripts to make the overall process substantially easier (i.e. less typing to do ☺).

Start OC4J

Open a new Command Prompt window and execute the following command sequence:

```
> cd /d C:\OC4JWS\Haikus
> bin\oc4j.cmd
```

When OC4J is up and running, the Command Prompt window should resemble the following screen:

You will need to keep OC4J running throughout this exercise.

Build the Web Service

Open a new Command Prompt window and execute the following sequence:

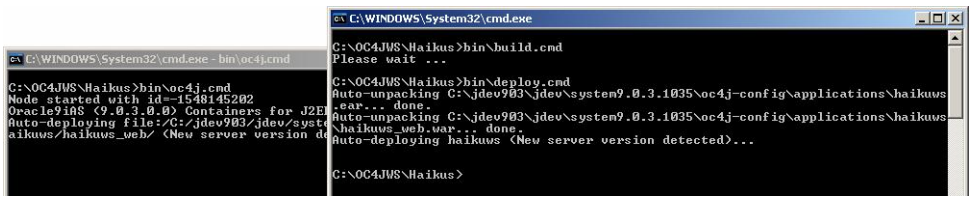
```
> cd /d C:\OC4JWS\Haikus
> bin\build.cmd
```

Deploy the Web Service

In the same Command Prompt window issue the following command:

```
> bin\deploy.cmd
```

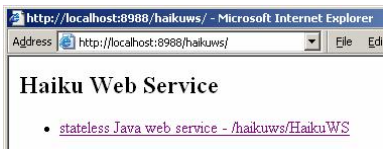
You should now see both Command Prompt windows reflect the new system state, as shown here:



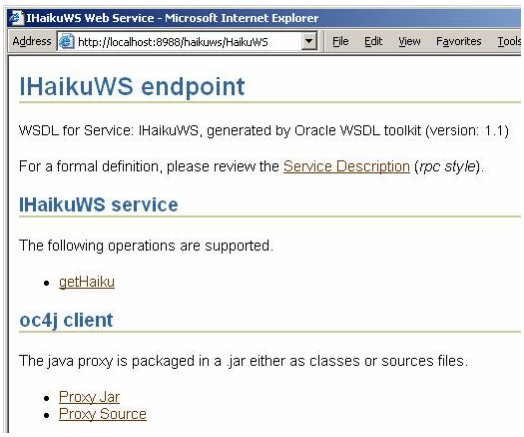
Investigate the Deployed Service

Open a new web browser window and point it to http://localhost:8988/haikuws/.

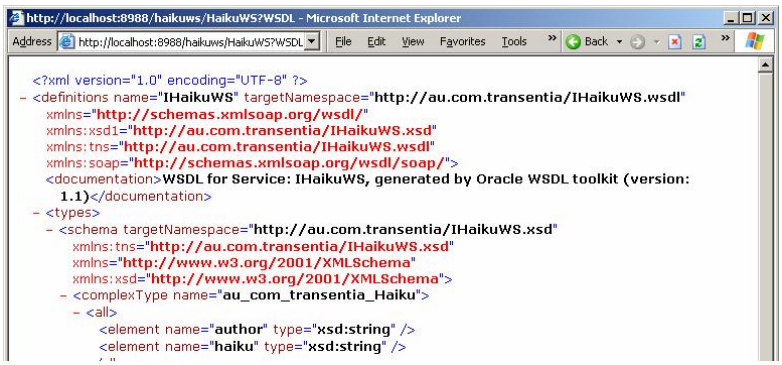
You will see an automatically generated page containing a single link:



Click on the "stateless Java web service - /haikuws/HaikuWS" link to be taken to the "IHaikuWS endpoint" page.



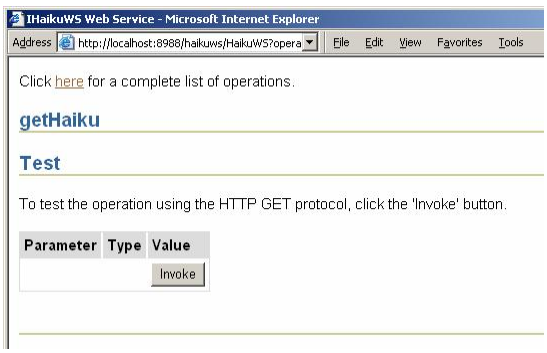
From this page, it is possible to examine the WSDL Service Description file that OC4J automatically. **Follow the Service Description link** to be presented with a listing of the WSDL file similar to this:



Use the browser's Back button to return to the previous page.

The "IHaikuWS endpoint" page also provides a link for each method that is a part of the Web Service's interface.

Follow the getHaiku link. You will be taken to a page resembling the following screenshot.



Click on the Invoke button. This will invoke the interface function and result in a new window that displays the SOAP response message as shown below:

The "IHaikuWS endpoint" page also provides links to allow you to download auto-generated code (source and binary Java Archive) that may make the task of writing client applications easier.

Custom Serialization

In the previous section, you built a simple WebService that relied on the SOAP infrastructure's standard behaviour for serialization of data across the network (for this simple example, the standard infrastructure is actually sufficient…but it is good to look at more in-depth issues). In this section you will augment that service to use custom serialization and provide a 'proper' client application that will show how to programmatically access the augmented WebService.

Create the Custom Serializer for the Haku JavaBean

To illustrate the mechanisms SOAP possesses for dealing with complex datatypes, this session will create and use a custom serializer to "hand craft" the "on-the-wire" representation of a Haiku JavaBean instance. This is probably the hardest aspect of using SOAP.

Although not strictly required in this situation, it is useful to know how this can be done.

Edit the file **C:\OC4JWS\Haikus\src\transentia\HaikuSerializer.java**. The content should be as follows:

```
package transentia;

import java.io.*;
import org.apache.soap.encoding.soapenc.*;
import org.apache.soap.rpc.*;
import org.apache.soap.util.*;
import org.apache.soap.util.xml.*;
import org.w3c.dom.*;

public class HaikuSerializer
  implements Serializer, Deserializer
  {
  private final String
    htag = "HAIKU_DATA",
    atag = "AUTHOR_DATA";
```

```
public void marshall(String inScopeEncStyle, Class javaType,
                     Object src, Object context, Writer sink,
                     NSStack nsStack,
                     XMLJavaMappingRegistry xjmr,
                     SOAPContext ctx)
  throws IllegalArgumentException, IOException
  {
  nsStack.pushScope();

  SoapEncUtils.generateStructureHeader
    (inScopeEncStyle, javaType, context, sink, nsStack, xjmr);
  sink.write(StringUtils.lineSeparator);

  Haiku h = (Haiku)src;

  Parameter param =
    new Parameter(htag, java.lang.String.class,
                  h.getHaiku(), null);
  xjmr.marshall(inScopeEncStyle, Parameter.class, param, null,
                sink, nsStack, ctx);
  sink.write(StringUtils.lineSeparator);

  param = new Parameter(atag, java.lang.String.class,
                        h.getAuthor(), null);
  xjmr.marshall(inScopeEncStyle, Parameter.class, param, null,
                sink, nsStack, ctx);
  sink.write(StringUtils.lineSeparator);

  sink.write("</" + context + '>');

  nsStack.popScope();
  }

public Bean unmarshall(String inScopeEncStyle,
                       QName elementType,
                       Node src, XMLJavaMappingRegistry xjmr,
                       SOAPContext ctx)
  throws IllegalArgumentException
  {
  Element root = (Element)src;
  Element tempEl = DOMUtils.getFirstChildElement(root);
  Haiku h;

  try
    {
    h = (Haiku)Haiku.class.newInstance ();
    }
  catch (Exception e)
    {
    throw new IllegalArgumentException
      ("Problem instantiating bean: " + e.getMessage());
    }

  while (tempEl != null)
    {
    Bean paramBean = xjmr.unmarshall
      (inScopeEncStyle, RPCConstants.Q_ELEM_PARAMETER,
       tempEl, ctx);
    Parameter param = (Parameter)paramBean.value;
    String tagName = tempEl.getTagName();
```

```
      if (htag.equals(tagName))
        h.setHaiku ((java.lang.String)param.getValue());

      else if (atag.equals(tagName))
        h.setAuthor ((java.lang.String)param.getValue());
      tempEl = DOMUtils.getNextSiblingElement(tempEl);
      }

    return new Bean(Haiku.class, h);
    }
  }
```

Points to Note

- The marshall method is responsible for creating a fragment of XML representing the instance data of the Haiku JavaBean that is passed in to the method as the 'src' parameter. The unmarshall operation performs the inverse operation: parsing the received XML document fragment and instantiating a corresponding Haiku JavaBean instance.

Create the Java Client

Edit the file **C:\OC4JWS\Haikus\src\transentia\HaikuSerializer.java**. The content should be as follows:

```
package transentia.client;
import org.apache.soap.encoding.soapenc.*;
import org.apache.soap.encoding.*;
import org.apache.soap.util.xml.*;
import org.apache.soap.*;
import org.apache.soap.rpc.*;
import java.util.*;
import java.net.*;
import transentia.*;

public class HaikusClient
  {
  public static void main(String[] args)
    throws Exception
    {
    URL endpointURL =
      new URL("http://127.0.0.1:8988/haikuws/HaikuWS");
    Call call = new Call();
    call.setTargetObjectURI("transentia-IHaikuWS");
    call.setMethodName("getHaiku");

    call.setParams(new Vector());

    call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
    SOAPMappingRegistry smr = new SOAPMappingRegistry();
    HaikuSerializer hs = new HaikuSerializer();
    smr.mapTypes(Constants.NS_URI_SOAP_ENC,
                 new QName("http://transentia/IHaikuWS.xsd",
                           "transentia_Haiku"),
                 transentia.Haiku.class, hs, hs);
    call.setSOAPMappingRegistry(smr);
```

```
    Response response = call.invoke(endpointURL, "");
    String returnVal = null;

    if (response.generatedFault())
      {
      Fault fault = response.getFault();
      returnVal = fault.getFaultCode() + " " +
                  fault.getFaultString();
      }
    else
      {
      Parameter result = response.getReturnValue();
      returnVal = ((Haiku)result.getValue()).toString();
      }

    System.out.println(returnVal.toString());
    }
  }
```

Points to Note

- The client application takes care to register a custom serializer class for the transentia.Haiku JavaBean with its SOAPMappingRegistry object.

Create the Web Application Deployment Descriptor

The WebServicesAssembler tool automatically generates a default deployment descriptor for the WebService. Since this section of the exercise is all about modifying the SOAP system's default behaviour, you will need to create and install a deployment descriptor "by hand" so that the new behaviour is incorporated appropriately.

Edit the skeleton deployment descriptor file **C:\OC4JWS\Haikus\src\web.xml** to become as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app
  PUBLIC
    '-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN'
    'http://java.sun.com/dtd/web-app_2_3.dtd'>

<web-app>

<servlet>
  <servlet-name>
    stateless Java web service - /haikuws/HaikuWS
  </servlet-name>
  <servlet-class>
    oracle.j2ee.ws.StatelessJavaRpcWebService
  </servlet-class>
  <init-param>
    <param-name>class-name</param-name>
    <param-value>transentia.HaikuWS</param-value>
  </init-param>
  <init-param>
    <param-name>interface-name</param-name>
    <param-value>transentia.IHaikuWS</param-value>
  </init-param>
```

```
    <init-param>
      <param-name>custom-bean-qname</param-name>
      <param-value>
transentia.Haiku,http://transentia/IHaikuWS.xsd,transentia_Haiku
,transentia.HaikuSerializer,transentia.HaikuSerializer
      </param-value>
    </init-param>
</servlet>

<servlet-mapping>
  <servlet-name>
    stateless Java web service - /haikuws/HaikuWS
  </servlet-name>
  <url-pattern>/HaikuWS</url-pattern>
</servlet-mapping>

<welcome-file-list>
  <welcome-file>index.html</welcome-file>
</welcome-file-list>

</web-app>
```

Points to Note

- Note that the contents for <param-value> elements must be given on **one line only**. Any line breaks you see here are for formatting purposes *only*.
- Also note how the oracle.j2ee.ws.StatelessJavaRpcWebService servlet is configured to know that the class transentia.Haiku should be marshalled and unmarshalled by a custom serializer class: transentia.HaikuSerializer.

Build and Execute II

Note: ensure that your OC4J server is running before attempting this stage. If you have forgotten how to run the server, see the earlier section "Build and Execute I."

As before, you have been supplied with a number of build/execute scripts for the client application.

Build the new WebService

Open a new Command Prompt window and execute the following sequence:

```
> cd /d C:\OC4JWS\Haikus
> bin\undeploy.cmd
> bin\buildcustom.cmd
> bin\deploy.cmd
```

Note: you may want to briefly examine the **bin\buildcustom.cmd** file…it packages up the many steps necessary to unpack, modify and repackage the default ear file produced by the WebServicesAssembler tool. The script has a fair amount of work to do and so is quite complex.

Build the Client Application

Open a new Command Prompt window and execute the following command sequence:

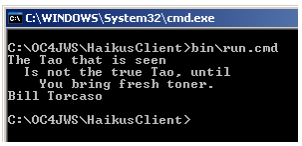
```
> cd /d C:\OC4JWS\HaikusClient
> bin\build.cmd
```

Run the Client Application

In the same Command Prompt window as before, Issue the command:

```
> bin\run.cmd
```

You should see a result similar to what is shown below:



Verification

To double-check that your custom serialization mechanism is actually being used, repeat the activities described in the earlier “Investigate the Deployed Service” section.

You should receive a reply message similar to the following:

Note the difference between the formatted data contained in the reply message you received earlier and the message that you have obtained this time.