IGATE Global Solutions Ltd

SOA & WebServices

Lab Guide

Table of Contents

LAB 1: INSTALLATION OF AXIS

Goals	Installation and setup of AXIS
Time	60 min
Lab Setup	Windows OS with JDK1.4 installed, Tomcat 5.0, Jakarta AXIS

INTRODUCTION

This Lab describes how to install Apache Axis. It assumes you already know how to write and run Java code and are familiar with XML. You should also have an application server or servlet engine and be familiar with operating and deploying to it. This Lab Guide assumes that you would be using Jakarta TOMCAT 5.0

STEP 0: CONCEPTS

Apache Axis is an Open Source SOAP server and client. SOAP is a mechanism for inter-application communication between systems written in arbitrary languages, across the Internet. SOAP usually exchanges messages over HTTP: the client POSTs a SOAP request, and receives either an HTTP error code or an HTTP success code and a SOAP response. Open Source means that you get the source, but that there is no formal support organization to help you when things go wrong.

Axis is implemented in the JAR file *axis.jar*; implementing the JAX-RPC API declared in the JAR files *jaxrpc.jar* and *saaj.jar*. It needs various helper libraries, for logging, WSDL processing and introspection. All these files can be packaged into a web application, *axis.war*, that can be dropped into a servlet container. Axis ships with some sample SOAP services. You can add your own by adding new compiled classes to the Axis webapp registering them.

Before you can do that, you have to install it and get it working.

STEP 1: PREPARING THE WEBAPP

Here we assume that you have a web server up and running on the localhost, at port 8080. If your server is on a different port, replace references to 8080 to your own port number.

In your Application Server installation, you should find a directory into which web applications ("webapps") are to be placed. Into this directory copy the webapps/axis directory from the xml-axis distribution. You can actually name this directory anything you want, just be aware that the name you choose will form the basis for the URL by which clients will access your service. The rest of this document assumes that the default webapp name, "axis" has been used; rename these references if appropriate.

STEP 2: SETTING UP THE LIBRARIES

In the Axis directory, you will find a WEB-INF sub-directory. This directory contains some basic configuration information, but can also be used to contain the dependencies and web services you wish to deploy.

Axis needs to have an XML parser in its classpath. If your application server or Java runtime does not make one visible to web applications, you need to download and add it. Java 1.4 includes the Crimson parser, so you *can* omit this stage, though the Axis team prefers Xerces.

To add an XML parser, acquire the JAXP 1.1 XML compliant parser of your choice. We recommend Xerces jars from the xml-xerces distribution, though others mostly work. Unless your JRE or app server has its own specific requirements, you can add the parser's libraries to axis/WEB-INF/lib.

STEP 3: STARTING THE WEB SERVER (TOMCAT)

STEP 4: VALIDATE THE INSTALLATION

After installing the web application and dependencies, you should make sure that the server is running the web application.

1.1.1 Look for the start page

Navigate to the start page of the webapp, usually http://localhost:8080/axis/, though of course the port may differ.
You should now see an Apache-Axis start page. If you do not, then the webapp is not actually installed, or the appserver is not running.

1.1.2 Validate Axis with happyaxis

Follow the link *Validate the local installation's configuration*
This will bring you to *happyaxis.jsp* a test page that verifies that needed and optional libraries are present. The URL for this will be something like http://localhost:8080/axis/happyaxis.jsp

If any of the needed libraries are missing, Axis will not work.
You must not proceed until all needed libraries can be found, and this validation page is happy.
Optional components are optional; install them as your need arises. If you see nothing but an internal server error and an exception trace, then you probably have multiple XML parsers on the classpath, and this is causing version confusion. Eliminate the extra parsers, restart the app server and try again.

1.1.3 Look for some services

From the start page, select *View the list of deployed Web services*. This will list all registered Web Services, unless the servlet is configured not to do so. On this page, You should be able to click on *(wsdl)* for each deployed Web service to make sure that your web service is up and running.

Note that the 'instant' JWS Web Services that Axis supports are not listed in this listing

1.1.4 Test a SOAP Endpoint

Now it's time to test a service. Although SOAP 1.1 uses HTTP POST to submit an XML request to the *endpoint*, Axis also supports a crude HTTP GET access mechanism, which is useful for testing. First let's retrieve the version of Axis from the version endpoint, calling the getVersion method: http://localhost:8080/axis/services/Version?method=getVersion This should return something like

```
 <?xml version="1.0" encoding="UTF-8" ?>
<soapenv:Envelope
 xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
   <getVersionResponse soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <getVersionReturn xsi:type="xsd:string">Apache Axis version: 1.1 Built on Jun 13,
  2003 (09:19:43 EDT)</getVersionReturn>
   <getVersionResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

The Axis version and build date may of course be different.

1.1.5 Test a JWS Endpoint

Now let's test a JWS web service. Axis' JWS Web Services are java files you save into the axis webapp *anywhere but the WEB-INF tree*, giving them the .jws extension. When some requests the .jws file by giving its URL, it is compiled and executed. The user guide covers JWS pages in detail.

To test the JWS service, we make a request against a built in example, EchoHeaders.jws (look for this in the axis directory).

Point your browser at http://localhost:8080/axis/EchoHeaders.jws?method=list .

This should return an XML listing of your application headers, such as

```
<?xml version="1.0" encoding="UTF-8" ?>
<soapenv:Envelope
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Body>
      <listResponse
         soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
         <listReturn xsi:type="soapenc:Array"
            soapenc:arrayType="xsd:string[6]"
            xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
            <item>accept:image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*</item>
            <item>accept-language:en-us</item>
            <item>accept-encoding:gzip, deflate</item>
            <item>user-agent:Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)</item>
            <item>host:localhost:8080</item>
            <item>connection:Keep-Alive</item>
         </listReturn>
      </listResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

Again, the exact return values will be different, and you may need to change URLs to correct host, port and webapp specifics.

STEP 5: INSTALLING NEW WEB SERVICES

So far you have got Axis installed and working, now it is time to add your own Web Service.

The process here boils down to (1) get the classes and libraries of this new service into the axis WAR directory tree, (2) tell the AxisEngine about the new file.

The first step is to add your code to the server.

In the WEB-INF directory, look for (or create) a "classes" directory (i.e. axis/WEB-INF/classes). In this directory, copy the compiled Java classes you wish to install, being careful to preserve the directory structure of the Java packages.

If your classes services are already packaged into JAR files, feel free to drop them into the WEB-INF/lib directory instead. Also add any third party libraries you depend on into the same directory.

After adding new classes or libraries to the Axis webapp directory, you must restart the webapp. This can be done by restarting your application server, or by using a server-specific mechanism to restart a specific webapp.

STEP 6: USE THE WEB SERVICE THROUGH THE CLIENT

LAB 2: LOGIN WEBSERVICE

Goal	Developing and using a Java Webservice
Time	90 Minutes
Lab Setup	Windows OS with JDK1.4 installed, Eclipse, Tomcat 5.0, Jakarta AXIS

Problem Statement:

Develop a LOGIN webservice that provides the methods viz. validate(), addUser(), changePassword( ) & deleteUser( ). Use a Arraylist to maintain the list of all valid users.
Develop a LoginSerclient.java to test the Login Webservice. It should allow the user to enter the user name & the password and use the web service for validating it. It should also allow the functionalities like register new user, change password and delete user.

Step -1: Code the LoginService class which will have the validate ( ), addUser ( ), changePassword ( ) & deleteUser ( )

```
package com.IGATE;
import java.util.HashMap;

/*
 *
 * LoginService - WebService
 *
 */

public class LoginService
{

        HashMap userlist = new HashMap();

        public LoginService()
        {
                //TO DO: put the valid users in the hashMap
        }

        public boolean validate(String username,String password)
        {
                boolean valid=false;

                /* TO DO:

                  Validate the username and password against the valid user list in
                  hashmap and set the "valid" Boolean variable accordingly

                */

                return valid;
        }

        public boolean addUser(String username, String password)
        {

                /* TO DO:
       Search for the username in the userlist and if it exists
       return an error.
     */

                userlist.put(username,password);
                return true;
        }

        public String changeUser(String username, String oldPassword, String newPassword)
        {
                String status = "Password changed successfully";

                /* TO DO:
       Search for the username in the userlist and valid the
       oldPassword. If valid then change from oldPassword to newPassword
     */

                return status;
        }

        public boolean deleteUser(String username)
        {
                boolean success = true;
                if(userlist.containsKey(username))
                {
                        userlist.remove(username);
                }
                else
                {
                        success = false;
                }
                return success;
        }


}
```

Compile the LoginService class

Step -2: Host the LoginService in the web server (i.e. copy the LoginService.class in the WEB-INF\classes folder in “axis” webapp)

Step - 3: Develop the client (LoginSerClient.java) to test the LoginService

```
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import javax.xml.namespace.QName;

public class LoginSerClient
{
 public static void main(String[] args)
 {
        try{
                String endpoint="http://localhost:8080/axis/services/LoginService";
                Service  service = new Service();
                Call     call    = (Call) service.createCall();
      call.setTargetEndpointAddress(new java.net.URL(endpoint) );
                call.setOperationName(new QName("", "validate"));

      Boolean valid = (Boolean)call.invoke(new
                  Object[]{"testuser","pwd"});

                if(valid.booleanValue())
                {
                        System.out.println("Valid user");
                }else{
                        System.out.println("InValid user");
        }

     } catch (Exception e) { System.err.println(e.toString());}
        } //end of main()
}//end of class
```

Step - 4: Compile and run LoginSerClient class
The following output will be displayed:

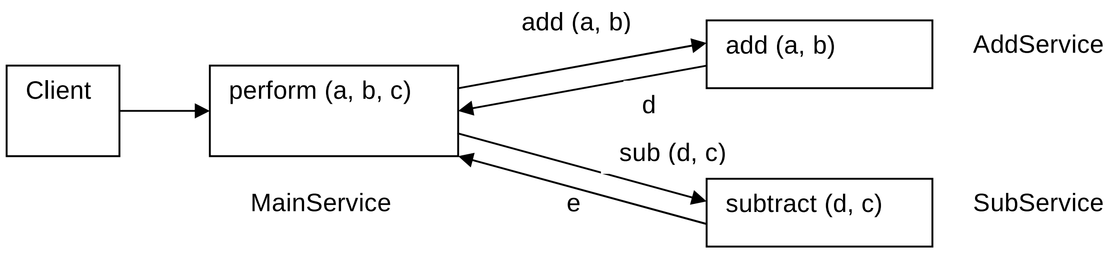
```
Valid User
```

Step 5: Test the other methods (viz addUser(), changeUser() and deleteUser())of the Login Web Service

LAB 3: INTEGRATING WEBSERVICES

Goal	Communication between Web Services
Time	90 Minutes
Lab Setup	Windows OS with JDK1.4 installed, Tomcat 5.0, Jakarta AXIS

Problem Statement: Develop three web services viz. – MainService, AddService and SubService. The interaction between the web services should be as shown in the following figure:



The MainService has perform (a, b, c) method, the AddService has add (a, b) method and SubService has subtract (d, c) method. Develop a client.jsp page which calls the perform method of MainService which in turn calls the add method of AddService to perform the addition and return the result (d) to the MainService perform method. Perform method of MainService will then call the subtract method of SubService to subtract "c" from the result (d) returned from AddService. The final result (e) returned by SubService to MainService will returned to the client. The client should display this result.

Step 1: Code for the AddService Web service and save the file as AddService.jws

```
public class AddService {
 public int add(int i1, int i2)
 {
  return i1 + i2;
 }

}
```

Step 2: Code for the SubService Web service and save the file as SubService.jws

```
public class SubService {

 public int subtract(int i1, int i2)
 {
  return i1 - i2;
 }
}
```

Step 3: Code for the MainService Web service and save the file as MainService.jws

```
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import javax.xml.namespace.QName;


public class MainService {
  public int perform(int i1, int i2, int i3)
  {
          Integer subresult = null;
           try {
                    String endpoint = "http://localhost:8080/axis/AddService.jws";
        Service  serviceadd = new Service();
                    Call     calladd    = (Call) serviceadd.createCall();

                    calladd.setTargetEndpointAddress( new java.net.URL(endpoint) );
                    calladd.setOperationName(new QName("", "add"));


       Integer addresult = (Integer) calladd.invoke( new Object[] { new
                          Integer(i1), new Integer(i2)} );

                   endpoint = "http://localhost:8080/axis/SubService.jws";

                   Service  servicesub = new Service();
                   Call     callsub   = (Call) servicesub.createCall();

                   callsub.setTargetEndpointAddress( new java.net.URL(endpoint) );
                   callsub.setOperationName(new QName("", "subtract"));

                   subresult = (Integer) callsub.invoke( new Object[] { addresult, new
                                       Integer(i3)} );

                   } catch (Exception e) {
                            System.err.println(e.toString());
                           /* TO DO:
            Handle the error conditions correctly. 'subresult' can be
            null and return statement will give an exception.
            Instead the MainService should return an error to the caller.
          */
                   }
                   return subresult.intValue();

  }//end of perform

 }//end of class
```

Step 4: Code for the Client (TestMainService.java) to test these Web service.

```
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import javax.xml.namespace.QName;

public class TestMainService
{
        public static void main(String[] args)
        {
                try {
                        String endpoint =

                        "http://localhost:8080/axis/MainService.jws";

                         Service  service = new Service();
                         Call     call    = (Call) service.createCall();

                         call.setTargetEndpointAddress( new java.net.URL(endpoint) );
                         call.setOperationName(new QName("", "perform"));

                         Integer ret = (Integer) call.invoke( new Object[] { new
                Integer(5), new Integer(10), new Integer(7)} );

                    System.out.println("Result="+ret);

                 } catch (Exception e) { System.err.println(e.toString());}
        }//end of main

}//end of class
```

Step 5: Compile TestMainService class and run it. The output should be as follows:

```
Result=8
```

LAB 4: USING WSDL2JAVA

Goal	Using WSDL2Java
Time	90 Minutes
Lab Setup	Windows OS with JDK1.4 installed, Eclipse, Tomcat 5.0, Jakarta AXIS

Problem Statement: Create & deploy a "Bonus" Web service which has one method – getBonus (Employee emp). Use WSDL2Java for getting bonus by sending employee object to the web service

Step 1: Code for the employee class with 3 properties viz. name, age & salary. Define getter and setter methods for all the 3 properties. The employee class must implement java.io.Serializable interface. Compile the employee class and place it in **axis\WEB-INF\classes** folder.

Step 2: Code for the Web Service BonusService as follows:

```
package com.IGATE;
import DefaultNamespace.Employee;

public class BonusService
{
        public double getBonus(Employee emp)
         {

                final double MAXLIMIT = 6000.00;
                double bonus = emp.getSalary()* 0.088;
                if(bonus>MAXLIMIT)
                        return MAXLIMIT;
                else
                        return bonus;
         }
}
```

Step 3 Create the following wsdd file (deployBS.wsdd) for deploying the BonusService:

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
      xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

 <service name="BonusService" provider="java:RPC">
   <parameter name="className" value="com.IGATE.BonusService"/>
   <parameter name="allowedMethods" value="*"/>
   <beanMapping qname="myNS:Employee" xmlns:myNS="urn:DefaultNamespace"
    languageSpecificType="java:DefaultNamespace.Employee"/>
 </service>
```

```
</deployment>
```

Step 4: Deploy the BonusService using wsdd file

Run> org.apache.axis.client.AdminClient deployBS.wsdd

If the Web Service is successfully deployed then you would get the following output:

```
Processing file deployBS.wsdd
<Admin>Done processing</Admin>
```

Step 4: Generate the java classes from the deployed Web service from its WSDL by using WSDL2Java utility.

Run> org.apache.axis.wsdl.WSDL2Java   http://localhost:8080/axis/services/BonusService?WSDL

This will generate the Java classes from the BonusService. These can be used by the client to consume this Web service.

Step 5: Code the client class (GetBonusClient.java) to consume the BonusService

```
package com.IGATE;

import java.rmi.RemoteException;
import javax.xml.rpc.ServiceException;
import DefaultNamespace.Employee;
import localhost.axis.services.BonusService.*;

public class GetBonusClient
{
 public static void main(String[] args)
 {
        Employee emp = new Employee();
        emp.setName("Ajay");
        emp.setAge("33");
        emp.setSalary(50000.00);


        BonusServiceService service = new BonusServiceServiceLocator();
        try
        {
        localhost.axis.services.BonusService.BonusService bonusService =
                        service.getBonusService();
         try
         {
                System.out.println("Bonus is : " +   bonusService.getBonus(emp));
         }
         catch (RemoteException e1) { e1.printStackTrace();          }

        }
        catch (ServiceException e){ e.printStackTrace(); }
```

```
    }//end of main

} //end of class
```

Step 6: Compile and run the GetBonusClient class. The output should be as follows:

```
Bonus is : 4400
```

LAB 5: USING JAXM

Goal	Using JAXM
Time	120 Minutes
Lab Setup	Windows OS with JDK1.4 installed, Eclipse, Tomcat 5.0, JAXM

Problem Statement: Develop three servlets (S1, S2 & S3). S1 is a normal servlet while S2 & S3 are JAXM servlets. S1 should create the following soap message & send it to S2 servlet:

```
<Body>
        <add  x = “3”   y = “5”/>
</Body>
```

S2 servlet should include the <sub> tag to the message as follows and send it to S3 servlet:

```
<Body>
        <add x = “3”  y = “5” />
        <sub x = “10”  y = “3” />
</Body>
```

S3 servlet must extract the soap message & performs addition & subtraction & generates the following SOAP reply message :

```
<Body>
        <Results>
                <addResult>8< /addResult>
                <subResult>7</subResult>
        </Results>
</Body>
```

Access the S1 servlet through a JSP page (a click button) and display the addresult & subresult on the same JSP page.

Step 1: Create a new webapp “JAXM-simple” along with its directory structure (i.e. WEB-INF\classes, lib, etc.)

Step 2: Place the following jar files in the JAXM-simple\WEB-INF\lib folder**:**

activation.jar
dom4j.jar
jaxm-api.jar
jaxm-runtime.jar
mailapi.jar
saaj-api.jar
saaj-impl.jar
soap.jar

Step 3: Code S2 JAXM Servlet as follows:

```
package com.IGATE;

import java.io.FileOutputStream;
import java.net.URL;
import javax.xml.soap.*;
import javax.xml.messaging.*;
import javax.servlet.*;


public class S2Servlet      extends JAXMServlet      implements ReqRespListener
{
        static MessageFactory fac = null;
        String to=null;
        //          Connection to send messages.
        private SOAPConnection con;

        static {
                try {
                        fac = MessageFactory.newInstance();
                } catch (Exception ex) {
                        ex.printStackTrace();
                }
        };

        public void init(ServletConfig servletConfig) throws ServletException {
                super.init(servletConfig);
                try {
                                SOAPConnectionFactory scf =
                    SOAPConnectionFactory.newInstance();
                                        con = scf.createConnection();
                        } catch(Exception e) {
                          System.out.println("Unable to open a SOAPConnection"+ e);
                        }
}

        // This is the application code for handling the message. Once the
        // message is received the application can retrieve the soap part, the
        // attachment part if there are any, or any other information from the
        // message.

        public SOAPMessage onMessage(SOAPMessage message) {
                System.out.println("On message called in receiving servlet");
                String retval ="<html> <H4>";
                SOAPMessage reply = null;
                try {
                        System.out.println("Here's the message: ");
                        message.writeTo(System.out);


                        //TO DO: Retrieve the message body & add the "sub" element
                        StringBuffer urlSB=new StringBuffer(
                    "http://localhost:8080/jaxm-simple");

                        String reqBase=urlSB.toString();

                        // Create an endpoint for the recipient of the message.
                        if(to==null) {
                                to=reqBase + "/S3Servlet";
                        }

                        URL urlEndpoint = new URL(to);


                        System.err.println("Sending message to URL: "+urlEndpoint);
                        System.err.println("Sent message is logged in \"sent2.msg\"");

                        retval += " Sent message (check \"sent2.msg\") and ";

                        FileOutputStream sentFile = new FileOutputStream("sent2.msg");
                        message.writeTo(sentFile);
                        sentFile.close();

                        // Send the message to the provider using the connection.
                        reply = con.call(message, urlEndpoint);

                        if (reply != null) {
                                FileOutputStream replyFile = new
                       FileOutputStream("reply2.msg");
                                reply.writeTo(replyFile);
                                replyFile.close();
                                System.err.println("Reply logged in \"reply2.msg\"");
                                retval+= "received reply (check \"reply2.msg\")</H4></html>";

                        } else {
                                System.err.println("No reply");
                                retval += " no reply was received. </H4> </html>";
                        }

                } catch(Throwable e) {
                        e.printStackTrace();
                        System.out.println("Error in constructing or sending message " +
                   e.getMessage());
                        retval += " There was an error " + "in constructing or sending
             message. </H4> </html>";
                }
                System.out.println(retval);
                return reply;
        }// end of onMessage
}//end of class
```

Step 4: Code S3 JAXM Servlet as follows:

```
package com.IGATE;

import java.util.Iterator;
import javax.xml.soap.*;
import javax.xml.messaging.*;
import javax.servlet.*;

public class S3Servlet      extends JAXMServlet      implements ReqRespListener
{
        static MessageFactory fac = null;
        String to=null;
        //      Connection to send messages.
        private SOAPConnection con;

        static {
                try {
                        fac = MessageFactory.newInstance();
                } catch (Exception ex) {
                        ex.printStackTrace();
                }
        };

        public void init(ServletConfig servletConfig) throws ServletException {
                super.init(servletConfig);

                // Initialize the connection 'con' if it will be used by onMessage.
        }

        // This is the application code for handling the message. Once the
        // message is received the application can retrieve the soap part, the
        // attachment part if there are any, or any other information from the
        // message.

        public SOAPMessage onMessage(SOAPMessage message)
   {
          /* TO DO : Retrieve the message body & perform the addition and subtraction
      using the retrieved values. Then generate the return message as mentioned
      in the problem statement. */

                return message;
        }
}
```

Step 5: Code for the S1 Servlet as follows:

```
package com.IGATE;

import java.io.*;
import java.net.URL;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.xml.soap.*;

public class S1Servlet extends HttpServlet
```

```
{
        String to = null;
        String data = null;
        ServletContext servletContext;

        // Connection to send messages.
        private SOAPConnection con;

        public void init(ServletConfig servletConfig) throws ServletException {
                super.init( servletConfig );
                servletContext = servletConfig.getServletContext();

                try {
                SOAPConnectionFactory scf = SOAPConnectionFactory.newInstance();
                        con = scf.createConnection();
                } catch(Exception e) {
                        System.out.println("Unable to open a SOAPConnection"+ e);
                }

        }

        public void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException {

                String retval ="<html> <H4>";

                try {
                        //TO DO: Create a message factory.

                        // TO DO: Create a message from the message factory.

                        /* TO DO: Message creation takes care of creating the SOAPPart - a
                          required part of the message as per the SOAP 1.1
                          specification. */

                        /* TO DO: Retrieve the envelope from the soap part to start
  building the soap message. */

                        // TO DO: Create a soap header from the envelope.

                        // Create a soap body from the envelope.
                        SOAPBody bdy = envelope.getBody();

                        // Add a soap body element to the soap body
                        SOAPBodyElement gltp
                        = bdy.addBodyElement(envelope.createName("add"));
                        gltp.addAttribute(envelope.createName("x"),"3");
                        gltp.addAttribute(envelope.createName("y"),"5");


                        StringBuffer urlSB=new StringBuffer();

 urlSB.append(req.getScheme()).append("://").
                       append(req.getServerName());
                        urlSB.append( ":" ).append( req.getServerPort() ).append(
                      req.getContextPath() );
                        String reqBase=urlSB.toString();

                        // Create an endpoint for the recipient of the message.
                        if(to==null) {
                                  to=reqBase + "/S2Servlet";
                        }

                        URL urlEndpoint = new URL(to);

                        System.err.println("Sending message to URL: "+urlEndpoint);
                        System.err.println("Sent message is logged in \"sent1.msg\"");

                        retval += " Sent message (check \"sent1.msg\") and ";

                        FileOutputStream sentFile = new FileOutputStream("sent1.msg");
                        msg.writeTo(sentFile);
                        sentFile.close();

                        // Send the message to the provider using the connection.
                        SOAPMessage reply = con.call(msg, urlEndpoint);

                        if (reply != null) {
                                  FileOutputStream replyFile = new
                          FileOutputStream("reply1.msg");
                                  reply.writeTo(replyFile);
                                  replyFile.close();
                                  System.err.println("Reply logged in \"reply1.msg\"");
                                  retval+="received reply(check \"reply1.msg\").</H4></html>";
                        } else {
                                  System.err.println("No reply");
                                  retval += " no reply was received. </H4> </html>";
                        }

                } catch(Throwable e) {
                         // TO DO: Indicate error in constructing or sending message
                }

                try {
                        OutputStream os = resp.getOutputStream();
                        os.write(retval.getBytes());
                        os.flush();
                        os.close();
                } catch (IOException e) {
                         // TO DO: Indicate error in sending servlet response
                }
        }

}
```

Step 6: Compile all the three servlets and place the class files in the JAXM-simple\WEB-INF\classes folder

Step 7: Make the <servlet> and the <servlet mapping> entries for all the three servlets in the web.xml file:

Step 8: Invoke S1Servlet through the browser as follows:
http://localhost:8080/JAXM-simple/S1Servlet

Step 9: The output of S1Servlet should be as follows:

Sent message (check "sent1.msg") and received reply (check "reply1.msg").

Check the messages of S1Servlet & S3Servlet in the sent1.msg & reply1.msg files.

LAB 6: PUBLISHING & QUERYING UDDI REGISTRY

Goal	Publishing & Querying a service using UDDI registry
Time	90 Minutes
Lab Setup	Windows OS with JDK1.5 installed, Eclipse, Tomcat 5.0, Jakarta AXIS

Problem Statement: Add & query organization & service on a UDDI registry

Step 1: Create the following two properties files:

```
#publish.properties file
#Registry Specific properties
query.url=http://<IP address of PC on which TOMCAT is running>:8080/RegistryServer/
publish.url=http://<IP address of PC on which TOMCAT is running>:8080/RegistryServer/
user.name=admin
user.password=admin

#if you are behind a firewall this needs to be configured
http.proxy.host=
http.proxy.port=
```

Save this file as "publish.properties" and place it in the same directory as the java class that will coded in step 2.

Step 2: Code for the java class (SaveOrganizationTest) to add or publish a Web Service to the UDDI registry running on the local or remote PC.

```
import javax.xml.registry.*;
import javax.xml.registry.infomodel.*;

import java.io.*;
import java.net.*;
import java.util.*;

public class SaveOrganizationTest {
```

```
String httpProxyHost = "";
String httpProxyPort = "";
String httpsProxyHost = "";
String httpsProxyPort = "";
String regUrli = "";
String regUrlp = "";
String username = "";
String password = "";
Properties connProps = new Properties();

private static final String QUERY_URL = "query.url";
private static final String PUBLISH_URL = "publish.url";
private static final String USER_NAME = "user.name";
private static final String USER_PASSWORD = "user.password";
private static final String PROXY_HOST = "http.proxy.host";
private static final String PROXY_PORT = "http.proxy.port";

public static void main(String[] args) {

  try {
    SaveOrganizationTest bqt = new SaveOrganizationTest();

                //Get publish.properties
                Properties properties = new Properties();
                properties.load(new FileInputStream("./publish.properties"));

    bqt.executeTest(properties);
  } catch (JAXRException e){
    System.out.println("FAILED" + e.getMessage());
  } catch (IOException ioe) {
                         System.out.println("Can not open properties file");
                 }
}

public void executeTest(Properties properties)
  throws JAXRException {

    try {
                                  assignUserProperties(properties);
      setConnectionProperties();

      ConnectionFactory factory = ConnectionFactory.newInstance();
      factory.setProperties(connProps);
      Connection conn = factory.createConnection();

      RegistryService rs = conn.getRegistryService();
      BusinessQueryManager bqm = rs.getBusinessQueryManager();
      BusinessLifeCycleManager blm = rs.getBusinessLifeCycleManager();

      PasswordAuthentication passwdAuth = new
            PasswordAuthentication(username, password.toCharArray());

      Set creds = new HashSet();
      creds.add(passwdAuth);
      conn.setCredentials(creds);
```

```
   Collection orgs = new ArrayList();

                  Organization org =
   blm.createOrganization(blm.createInternationalString("Solutions"));
   org.setDescription(blm.createInternationalString("Liberty and
                                    Freedom"));

   Service service =
   blm.createService(blm.createInternationalString("Federal Government
                                    Service"));
   service.setDescription(blm.createInternationalString("Services of the
                                  Federal Government"));

   ServiceBinding sb = blm.createServiceBinding();
  sb.setAccessURI("http://localhost:8080/axis/services/LoginService");
   service.addServiceBinding(sb);

   User user = blm.createUser();
   PersonName personName = blm.createPersonName("George Washington");

   org.setPrimaryContact(user);

   TelephoneNumber telephoneNumber = blm.createTelephoneNumber();
   telephoneNumber.setNumber("781-333-3333");
   telephoneNumber.setType(null);

   PostalAddress address
                  = blm.createPostalAddress("546789", "One USA Place", "Washington",
                     "DC", "USA", "02140", "");
               Collection postalAddresses = new ArrayList();
    postalAddresses.add(address);

               Collection emailAddresses = new ArrayList();
 EmailAddress emailAddress = blm.createEmailAddress("usaworks@usa.org");
   emailAddresses.add(emailAddress);

   Collection numbers = new ArrayList();
   numbers.add(telephoneNumber);

               user.setPersonName(personName);
   user.setPostalAddresses(postalAddresses);
   user.setEmailAddresses(emailAddresses);
   user.setTelephoneNumbers(numbers);

   //Concepts for NAICS and computer
   ClassificationScheme cScheme = blm.createClassificationScheme(
                                            blm.createInternationalString("ntis-gov:naics"),
                                            blm.createInternationalString(""));
   javax.xml.registry.infomodel.Key cKey =
                      (javax.xml.registry.infomodel.Key)
      blm.createKey("uuid:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2");
   cScheme.setKey(cKey);

   Classification classification = (Classification)
```

```
        blm.createClassification(cScheme,
                "Computer Systems Design and Related Services", "5415");

          org.addClassification(classification);

      ClassificationScheme cScheme1 = blm.createClassificationScheme(
                                        blm.createInternationalString("D-U-N-S"),
                                        blm.createInternationalString(""));
      javax.xml.registry.infomodel.Key cKey1 =
        (javax.xml.registry.infomodel.Key)
                blm.createKey("uuid:8609C81E-EE1F-4D5A-B202-3EB13AD01823");
      cScheme1.setKey(cKey1);

      ExternalIdentifier ei =
                blm.createExternalIdentifier(cScheme1, "D-U-N-S number",
                      "08-146-6849");

      org.addExternalIdentifier(ei);
      org.addService(service);

      orgs.add(org);

      BulkResponse br = blm.saveOrganizations(orgs);
      if (br.getStatus() == JAXRResponse.STATUS_SUCCESS) {
        System.out.println("Organization Saved");
      } else {
                System.err.println("One or more JAXRExceptions " +
                  "occurred during the save operation:");
                Collection exceptions = br.getExceptions();
                Iterator iter = exceptions.iterator();
                while (iter.hasNext()) {
                      Exception e = (Exception) iter.next();
                      System.err.println(e.toString());
                }
              }

    } catch (JAXRException e) {
      e.printStackTrace();
    }
}


       private void assignUserProperties(Properties props) {

              String proxyHost = ((String)props.get(PROXY_HOST)).trim();
              String proxyPort = ((String)props.get(PROXY_PORT)).trim();
              String queryURL = ((String)props.get(QUERY_URL)).trim();
              String publishURL = ((String)props.get(PUBLISH_URL)).trim();
              String user = ((String)props.get(USER_NAME)).trim();
              String pw = ((String)props.get(USER_PASSWORD)).trim();

              if (proxyHost != null){
                httpProxyHost = proxyHost;
                       httpsProxyHost = proxyHost;
              }
```

```
                if (proxyPort != null) {
                   httpProxyPort = proxyPort;
                           httpsProxyPort = proxyPort;
                }

                if (queryURL != null)
                   regUrli = queryURL;

                if (publishURL != null)
                   regUrlp = publishURL;

                if (user != null)
                   username = user;

                if (pw != null)
                   password = pw;

            }

      private void setConnectionProperties() {
         connProps.setProperty("javax.xml.registry.queryManagerURL",
                                         regUrli);
         connProps.setProperty("javax.xml.registry.lifeCycleManagerURL",
                                         regUrlp);
         connProps.setProperty("javax.xml.registry.factoryClass",
                                        "com.sun.xml.registry.uddi.ConnectionFactoryImpl");
              connProps.setProperty("com.sun.xml.registry.http.proxyHost",
                     httpProxyHost);
              connProps.setProperty("com.sun.xml.registry.http.proxyPort",
                     httpProxyPort);
              connProps.setProperty("com.sun.xml.registry.https.proxyHost",
                     httpsProxyHost);

        connProps.setProperty("com.sun.xml.registry.https.proxyPort",
                     httpsProxyPort);
      }
}//end of class
```

Step 3: Compile this class & run it. The output should be as follows:

```
            Organization Saved
```

Step 4: Code for a java class to query for a Web Service from the UDDI registry running on the local or remote PC.

```
import javax.xml.registry.*;
import javax.xml.registry.infomodel.*;

import java.io.*;
import java.util.*;

public class BusinessQueryTest {
   // edit these if behind firewall, otherwise leave blank
```

```
String httpProxyHost = "";
String httpProxyPort = "";
String regUrli  = "";
String regUrlp  = "";
Properties connProps = new Properties();

private static final String QUERY_URL = "query.url";
private static final String PUBLISH_URL = "publish.url";
private static final String PROXY_HOST = "http.proxy.host";
private static final String PROXY_PORT = "http.proxy.port";

public static void main(String[] args) {
  String company = "%Solu%";
  try {
    Properties properties = new Properties();
    properties.load(new FileInputStream("./query.properties"));
    BusinessQueryTest bqt = new BusinessQueryTest();
    bqt.executeQueryTest(properties, company);
  } catch (JAXRException e){
        System.err.println("Error during the test: " + e);
  } catch (IOException ioe) {
    System.err.println("Can not open properties file");
  }

}

public void executeQueryTest(Properties properties, String cname)
  throws JAXRException {
    try {
      assignUserProperties(properties);
      setConnectionProperties();

      ConnectionFactory factory = ConnectionFactory.newInstance();
      factory.setProperties(connProps);
      Connection conn = factory.createConnection();
      RegistryService rs = conn.getRegistryService();
      BusinessQueryManager bqm = rs.getBusinessQueryManager();


      ArrayList names = new ArrayList();
      names.add(cname);

      Collection fQualifiers = new ArrayList();
      fQualifiers.add(FindQualifier.SORT_BY_NAME_DESC);

      BulkResponse br = bqm.findOrganizations(fQualifiers,
        names, null, null, null, null);

      if (br.getStatus() == JAXRResponse.STATUS_SUCCESS) {
        System.out.println("Successfully queried the " +
          "registry for organization matching the " +
          "name pattern: \"" + cname + "\"");
        Collection orgs = br.getCollection();
        System.out.println("Results found: " + orgs.size() + "\n");
        Iterator iter = orgs.iterator();
```

```
            while (iter.hasNext()) {
              Organization org = (Organization) iter.next();
              System.out.println("Organization Name: " +
                getName(org));
              System.out.println("Organization Key: " +
                org.getKey().getId());
              System.out.println("Organization Description: " +
                getDescription(org));

              Collection services = org.getServices();
              Iterator siter = services.iterator();
              while (siter.hasNext()) {
                Service service = (Service) siter.next();
                System.out.println("\tService Name: " +
                  getName(service));
                System.out.println("\tService Key: " +
                  service.getKey().getId());
                System.out.println("\tService Description: " +
                  getDescription(service));
                Iterator sbs = service.getServiceBindings().iterator();
                while(sbs.hasNext()){
                ServiceBinding sb = (ServiceBinding)sbs.next();
                                     System.out.println("\tService Binding URL: " +
                sb.getAccessURI());}
                }

            }
          } else {
                    System.err.println("One or more JAXRExceptions " +
                      "occurred during the query operation:");
                    Collection exceptions = br.getExceptions();
                    Iterator iter = exceptions.iterator();
                    while (iter.hasNext()) {
                          Exception e = (Exception) iter.next();
                          System.err.println(e.toString());
                    }
                  }
        } catch (JAXRException e) {
          e.printStackTrace();
        }
    }

    private void assignUserProperties(Properties props) {
      String temp;

      temp = ((String)props.get(QUERY_URL)).trim();
      if (temp != null)
        regUrli = temp;

      temp = ((String)props.get(PUBLISH_URL)).trim();
      if (temp != null)
        regUrlp = temp;

      temp = ((String)props.get(PROXY_HOST)).trim();
      if (temp != null)
```

```
            httpProxyHost = temp;

        temp = ((String)props.get(PROXY_PORT)).trim();
        if (temp != null)
            httpProxyPort = temp;
    }

    private void setConnectionProperties() {
        connProps.setProperty("javax.xml.registry.queryManagerURL",
                                        regUrli);
        connProps.setProperty("javax.xml.registry.lifeCycleManagerURL",
                                        regUrlp);
        connProps.setProperty("javax.xml.registry.factoryClass",
                                        "com.sun.xml.registry.uddi.ConnectionFactoryImpl");
            connProps.setProperty("com.sun.xml.registry.http.proxyHost",
                    httpProxyHost);
            connProps.setProperty("com.sun.xml.registry.http.proxyPort",
                    httpProxyPort);
            connProps.setProperty("com.sun.xml.registry.https.proxyHost",
                    httpsProxyHost);

        connProps.setProperty("com.sun.xml.registry.https.proxyPort",
                    httpsProxyPort);

    private String getName(RegistryObject ro) throws JAXRException {
        try {
            return ro.getName().getValue();
        } catch (NullPointerException npe) {
            return "";
        }
    }

    private String getDescription(RegistryObject ro) throws JAXRException {
        try {
            return ro.getDescription().getValue();
        } catch (NullPointerException npe) {
            return "";
        }
    }
}
```

Step 5: Compile this class & run it. The output should display all the details of the Queried Web Service.