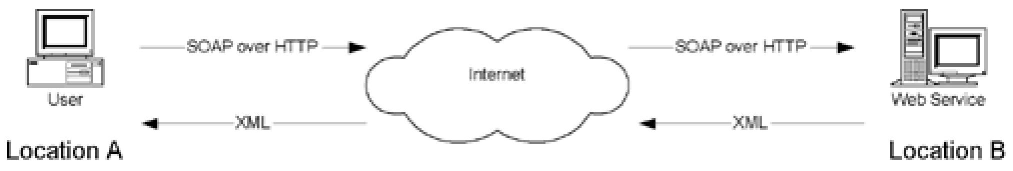
Table of Contents

Introduction: What is a Web Service?

Web Services can be described as a set of functionalities that are exposed over a network and are used as blocks for building distributed applications.

Although it is not a requirement, typically Web Services use Hypertext Transfer Protocol (HTTP) and SOAP (Simple Object Access Protocol) – mainly because:
- HTTP is a widely adopted protocol
- using HTTP is firewall friendly
- SOAP is a lightweight protocol
- SOAP is XML-based
- XML is platform and implementation independent
- XML is easy to create, parse and process

The image below presents the basic idea of how a Client invokes method executions on a Web Service:



In order for the Client to be aware of the semantics of the Web Service, the structure of the Web Service components is exposed using a Web Services Description Language (WSDL), which is a XML-based language. A client program that wants to connect to a web service first reads the WSDL file to determine what operations are available on the server. Any special datatypes used are described in the WSDL file in the form of XML Schema. The client can then use SOAP to actually call one of the operations listed in the WSDL file.

What is WS-Security?

Because of the nature of the internet, any form of communication is vulnerable to interception or modification. One way of securing requests and responses sent to and from a Web Service is to use SSL – however, because a Web Service can be a client of another web service, this type of protection does not provide end-to-end security.
WS-Security is a standard for adding security to SOAP Web Service message exchanges.
It uses a SOAP message-header element to attach the security information to messages, in the form of *tokens* conveying different types of claims (which can include names, identities, keys, groups, privileges etc.) along with encryption and digital-signature information. WS-Security supports multiple formats for tokens, multiple trust domains, multiple signature formats, and multiple encryption technologies, so in most cases the header information needs to include specific format and algorithm identification for each component.
WS-Security provides end-to-end security.

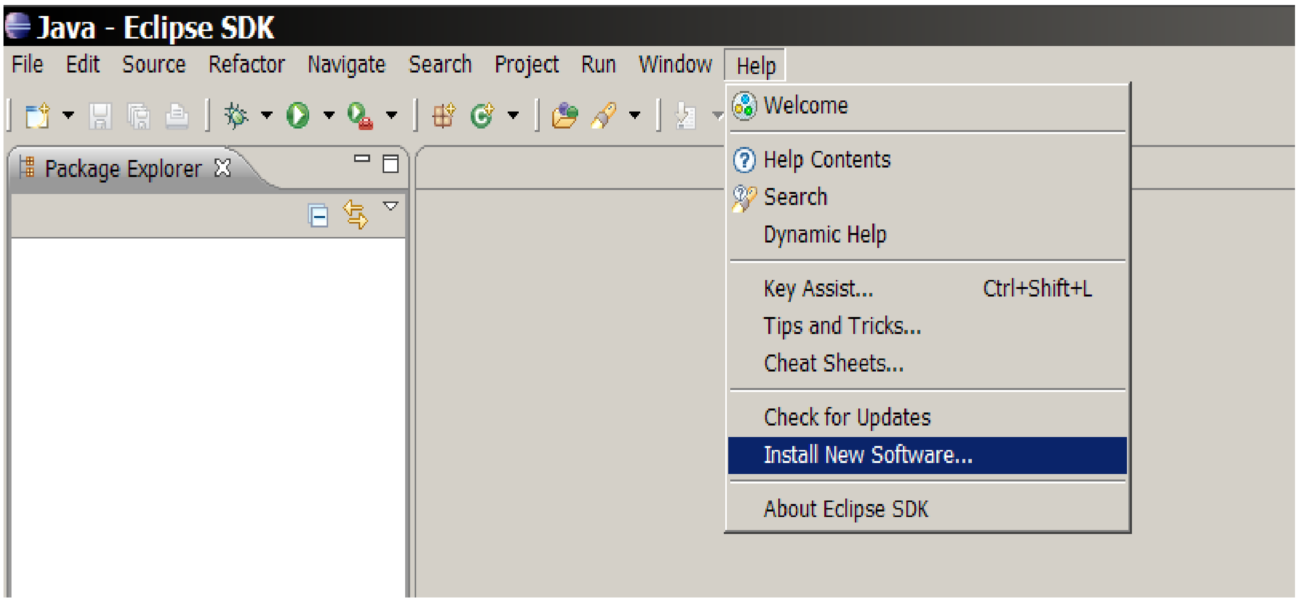
Installing required tools

Required tools:
Eclipse (http://www.eclipse.org/)
Apache Tomcat (http://tomcat.apache.org/)
Web Tools Platform (http://www.eclipse.org/webtools/)

Optional tools:
SoapUI (http://www.soapui.org/

Web Tools Platform Installation

Once Eclipse is installed/unpacked, choose from the upper bar:
help → install new software



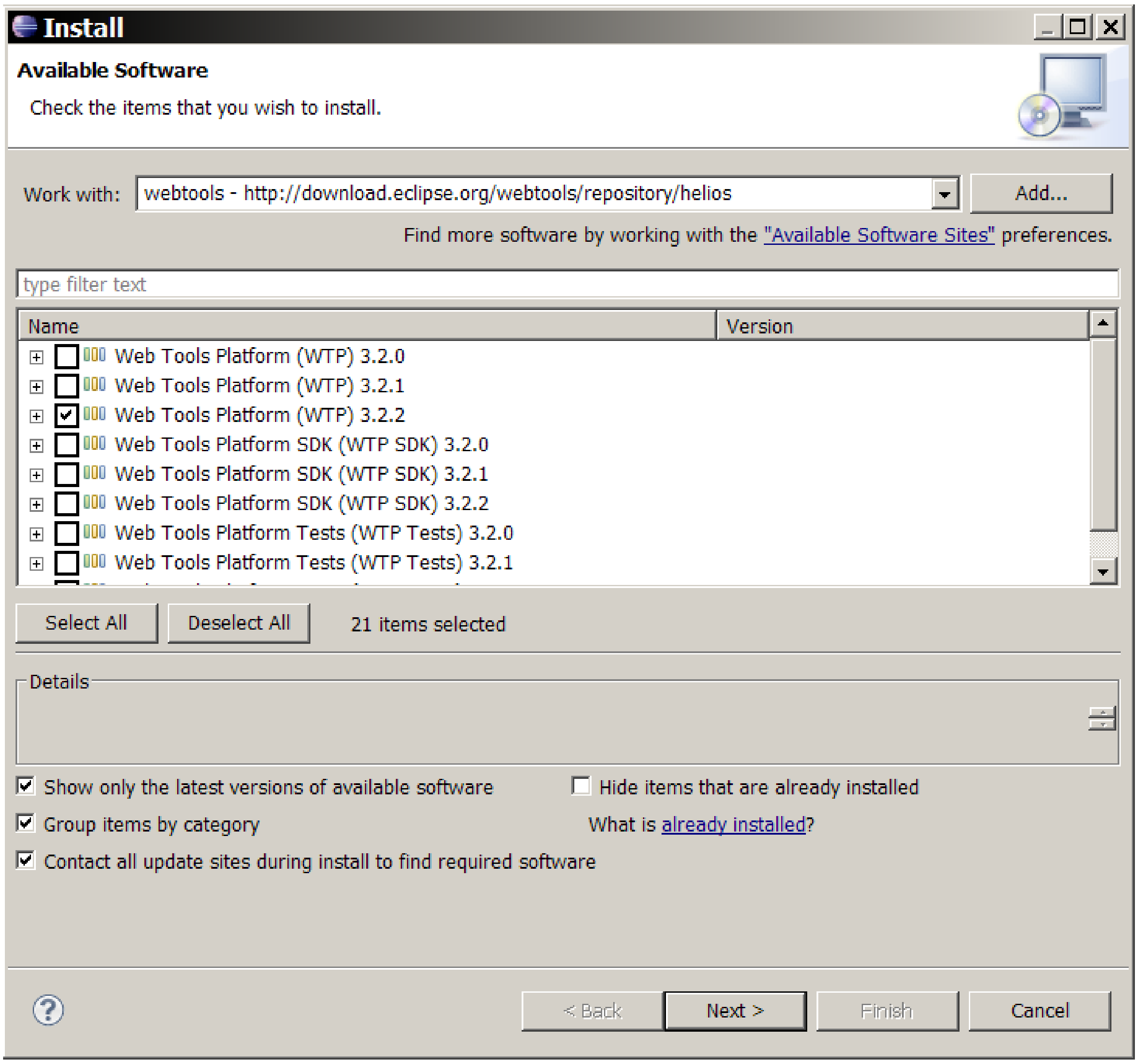
In the „work with" text-field insert: http://download.eclipse.org/webtools/repository/helios

Click "add"

Insert a name, for example „web tools"

Wait until the list of available items has loaded

Tick the most recent version of Web Tools Platform



Install

Available Software

Check the items that you wish to install.

Work with: webtools - http://download.eclipse.org/webtools/repository/helios

Find more software by working with the "Available Software Sites" preferences.

type filter text

Name	Version
☐ Web Tools Platform (WTP) 3.2.0	
☐ Web Tools Platform (WTP) 3.2.1	
☑ Web Tools Platform (WTP) 3.2.2	
☐ Web Tools Platform SDK (WTP SDK) 3.2.0	
☐ Web Tools Platform SDK (WTP SDK) 3.2.1	
☐ Web Tools Platform SDK (WTP SDK) 3.2.2	
☐ Web Tools Platform Tests (WTP Tests) 3.2.0	
☐ Web Tools Platform Tests (WTP Tests) 3.2.1	

21 items selected

Details

☑ Show only the latest versions of available software
☐ Hide items that are already installed
☑ Group items by category
What is already installed?
☑ Contact all update sites during install to find required software

Click „Next”

Click „Next”

Accept the terms of all necessary license agreements

Click “Finish”

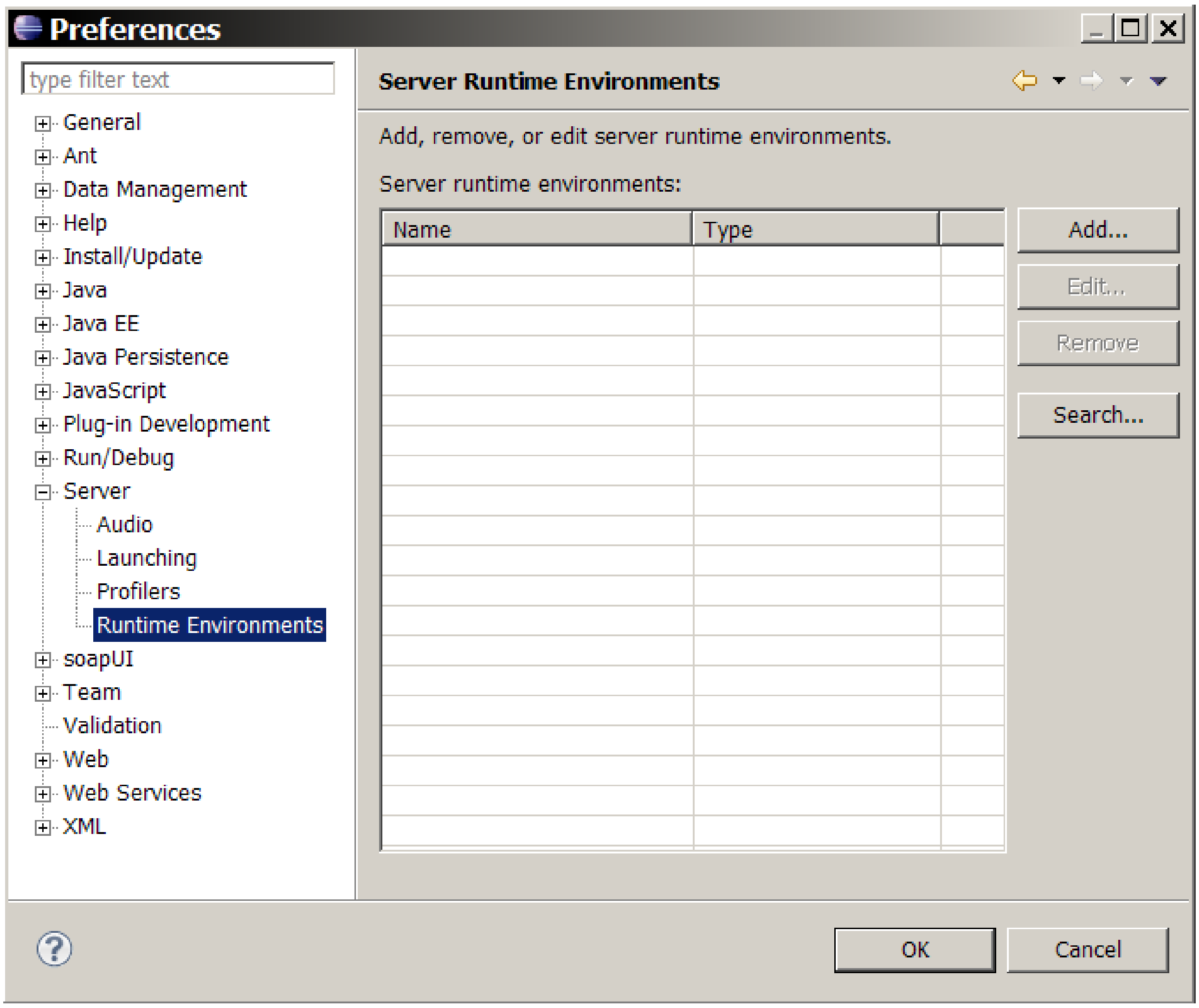
Wait until all software is installed.

When prompted choose to restart Eclipse.

Apache Tomcat Setup

Download the current version installation binaries of Apache Tomcat for the correct operating system from http://tomcat.apache.org and install normally.

Once the installation is complete, choose in Eclipse from the upper bar:
Window → preferences → server → runtime environments



Add, remove, or edit server runtime environments.

Server runtime environments:

Click „Add”

Choose the appropriate version of the Apache Tomcat server

Click „Next”

Specify the Tomcat installation directory

Click “Finish”

SoapUI Eclipse plug-in installation (optional)

From the upper bar in Eclipse choose:
help → install new software

In the „work with” text-field insert: http://www.soapui.org/eclipse/update

Click “Add”

Insert a name, for example „soapUI plug-in”

Wait until the list of all available items has loaded

Tick „soapUI”

Click “Next”

Click “Next”

Accept the terms of all necessary license agreements

Wait until all software is installed

Click “Finish”

When prompted choose to restart Eclipse

Creating a simple Web Service in Eclipse:

There are two approaches to creating a Web Service - bottom-up and top-down:

When creating a Web service using a bottom-up approach, first you create a Java bean or EJB bean and then use the Web services wizard to create the WSDL file and Web service.

When creating a Web service using a top-down approach, first you design the implementation of the Web service by creating a WSDL file. You can do this using the WSDL Editor. You can then use the Web services wizard to create the Web service and skeleton Java™ classes to which you can add the required code.

This example will show how to create a Web Service using the bottom-up method.

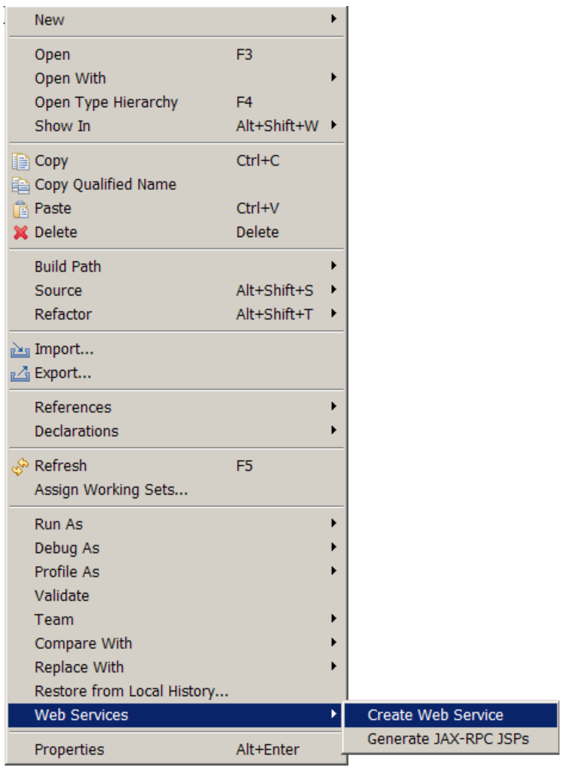
First, create a new Java Project in Eclipse.

Next, add new source files to the project, for example:

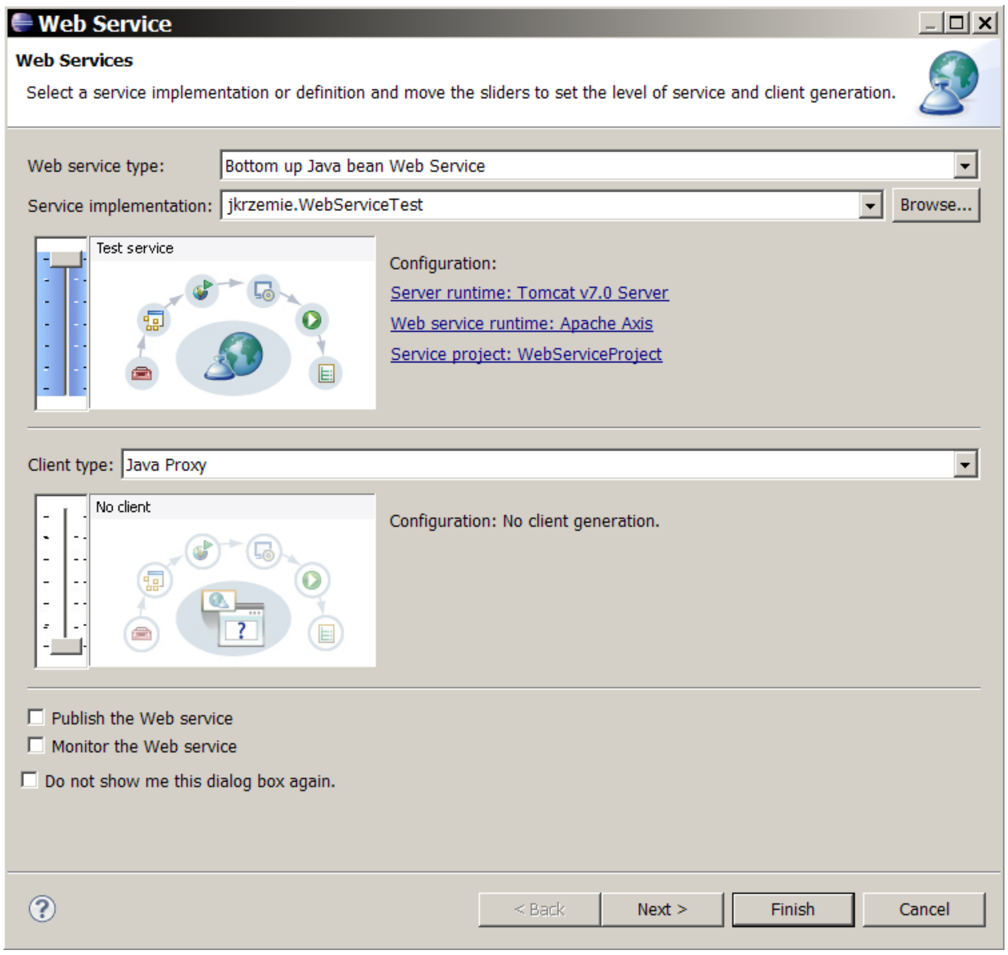
```
package jkrzemie.wstest;

public class TestClass
{
	public int x, y;
	public String description;

	public TestClass()
	{
		this.x = this.y = 2;
		this.description = "Description: ";
	}
}
```

```
package jkrzemie.wstest;

public class WebServiceTest
{


	public TestClass test(int x,int y, String d)
	{
		TestClass tc = new TestClass();
		tc.x *= x;
		tc.y *= y;
		tc.description += d;
		return tc;
	}
}
```

When the necessary code has been written, in the package explorer open the project, right click at WebServiceTest.java and select Web Services → Create Web Service



Bottom up Java bean Web Service should be selected.



Web Services

Select a service implementation or definition and move the sliders to set the level of service and client generation.

Web service type: Bottom up Java bean Web Service

Service implementation: jkrzemie.WebServiceTest

Configuration:
Server runtime: Tomcat v7.0 Server
Web service runtime: Apache Axis
Service project: WebServiceProject

Client type: Java Proxy

Configuration: No client generation.

☐ Publish the Web service
☐ Monitor the Web service
☐ Do not show me this dialog box again.

Remember to choose the correct service implementation.

The correct configuration should look like this:
    Server runtime: Tomcat Server
    Web service runtime: Apache Axis
    Service Project: current project

Set the left slider to: Start service

Click „next”

Select the correct method(s) from the list

Click „finish”

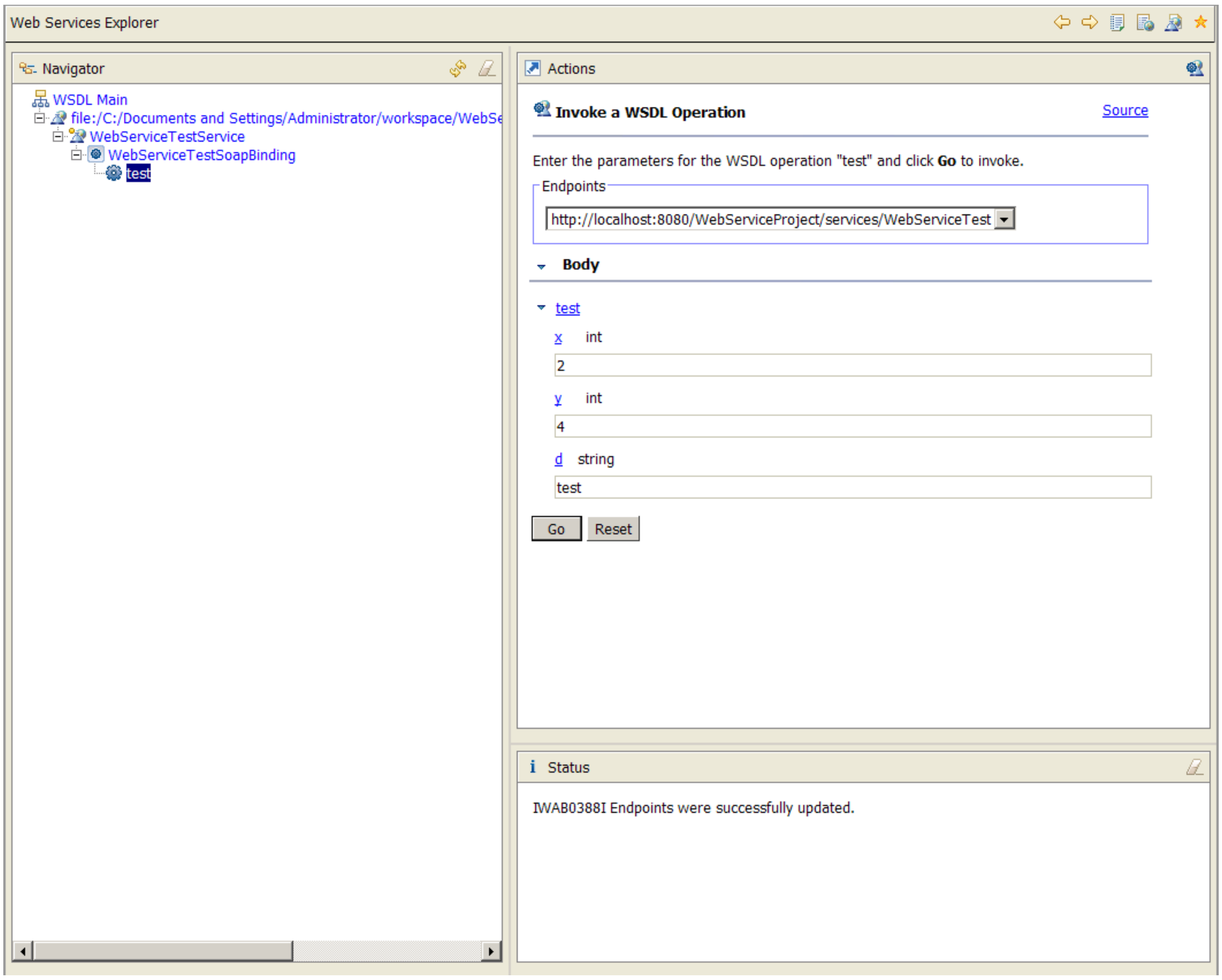
After a few seconds the Web Service should be up and running.

A new Web Services Explorer tab should automatically open in Eclipse.

Testing the created simple Web Service

If a tab with the Eclipse Web Service Explorer hasn't opened automatically, right-click on the WSDL file in package Explorer and select Web Services → Test with Web Services Explorer.

To test the service, click on the name of the operation that has been created, in the next window input values for all parameters and click „Go".



Web Services Explorer

Navigator

WSDL Main
file:/C:/Documents and Settings/Administrator/workspace/WebSe
WebServiceTestService
WebServiceTestSoapBinding
test

Actions

Invoke a WSDL Operation Source

Enter the parameters for the WSDL operation "test" and click **Go** to invoke.

Endpoints
http://localhost:8080/WebServiceProject/services/WebServiceTest

Body

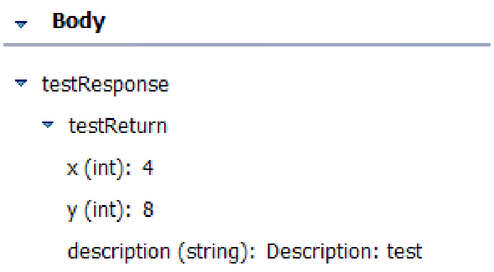
test
x int
2
y int
4
d string
test

Go Reset

Status

IWAB0388I Endpoints were successfully updated.

Check the response:

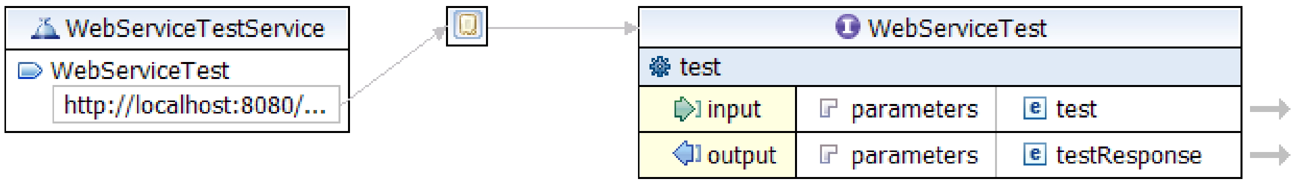


Body

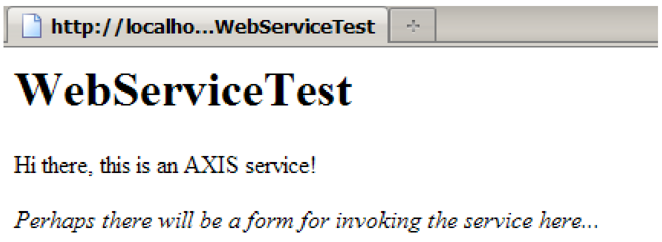
testResponse
testReturn
x (int): 4
y (int): 8
description (string): Description: test

Another way of checking if the Web Service is working is choosing in the package explorer the newly created WebServiceProject and going to WebContent → wsdl.

Next, open the wsdl file. The first element of the wsdl graph should have an address of the endpoint of the Web Service.



Open it in a web browser – you should see something similar to the screenshot below:



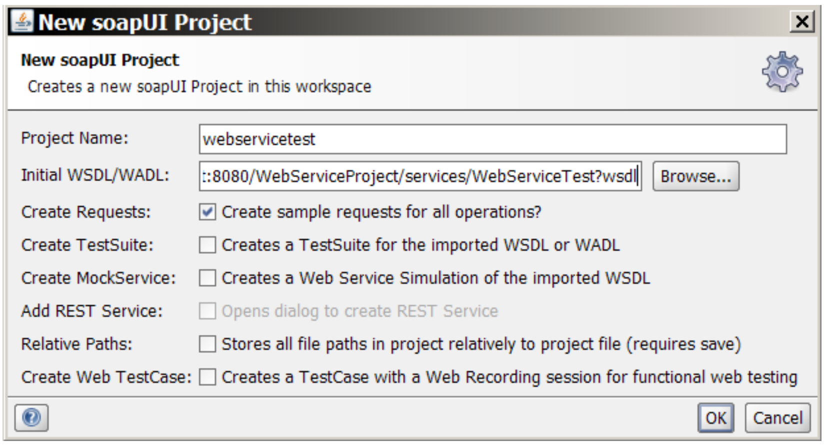
WebServiceTest

Hi there, this is an AXIS service!

Perhaps there will be a form for invoking the service here...

Testing the created simple Web Service with SoapUI

If you have installed SoapUI as a plug-in in Eclipse, switch to the SoapUI perspective.
Create a new SoapUI project, specifying the WSDL of the Web Service – the WSDL of the above example is available at: http://localhost:8080/WebServiceProject/services/WebServiceTest?wsdl
Leave all other options at their default values.

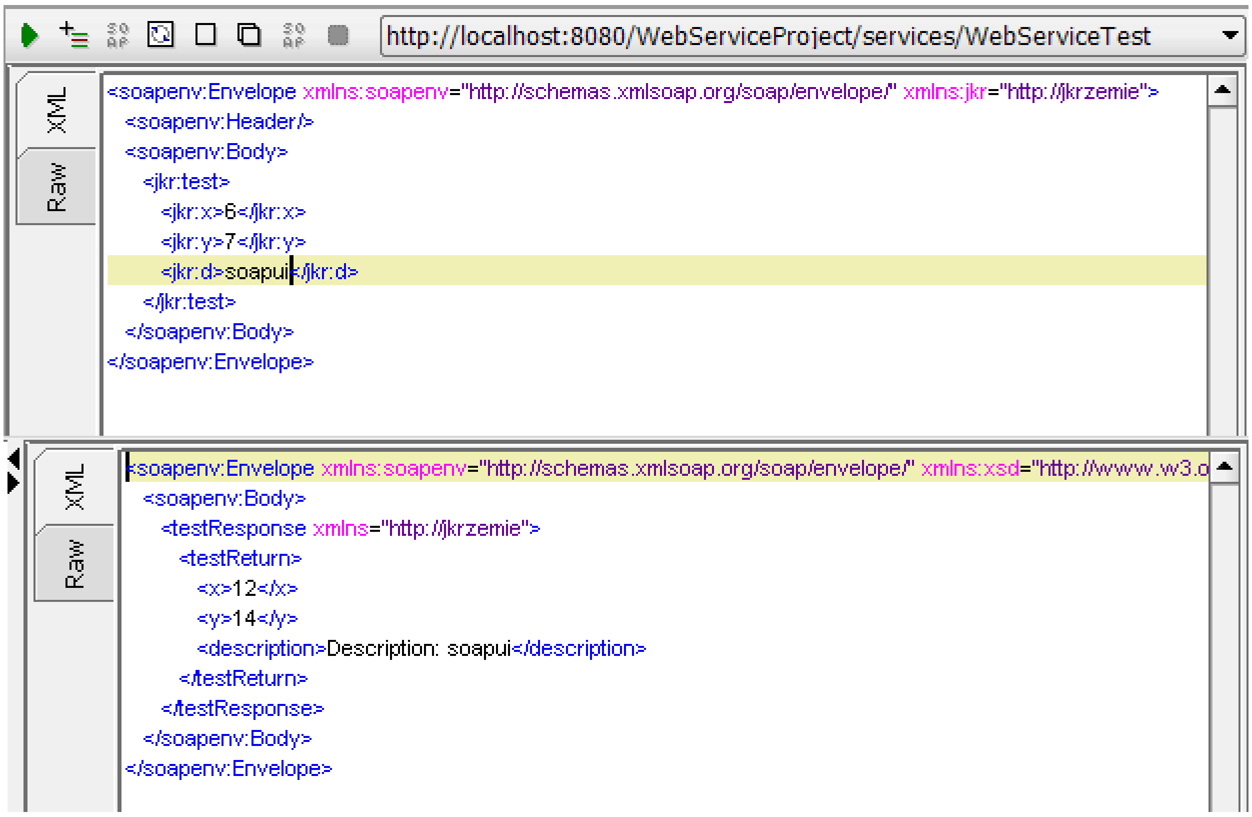

New soapUI Project

New soapUI Project
Creates a new soapUI Project in this workspace

Project Name:	webservicetest
Initial WSDL/WADL:	::8080/WebServiceProject/services/WebServiceTest?wsdl
Create Requests:	☑ Create sample requests for all operations?
Create TestSuite:	☐ Creates a TestSuite for the imported WSDL or WADL
Create MockService:	☐ Creates a Web Service Simulation of the imported WSDL
Add REST Service:	☐ Opens dialog to create REST Service
Relative Paths:	☐ Stores all file paths in project relatively to project file (requires save)
Create Web TestCase:	☐ Creates a TestCase with a Web Recording session for functional web testing

Once the project has been created, in the SoapUI navigator expand the project tree until reaching the methods of the Web Service.
Double click on the request of the method you want to test. SoapUI should propose a SOAP request message. Fill it in with appropriate values and click on the submit green arrow placed above the message window.

Check if the received response is as expected.


http://localhost:8080/WebServiceProject/services/WebServiceTest

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:jkr="http://jkrzemie">
   <soapenv:Header/>
   <soapenv:Body>
      <jkr:test>
         <jkr:x>6</jkr:x>
         <jkr:y>7</jkr:y>
         <jkr:d>soapui</jkr:d>
      </jkr:test>
   </soapenv:Body>
</soapenv:Envelope>
```

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.o
   <soapenv:Body>
      <testResponse xmlns="http://jkrzemie">
         <testReturn>
            <x>12</x>
            <y>14</y>
            <description>Description: soapui</description>
         </testReturn>
      </testResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

Creating a Web Service that runs via SSL

When running a Web Service via SSL, the process of creating the service itself remains the same – one has to ensure that the server that is running the Web Service provides connectivity over SSL.

Before reconfiguring Tomcat you must create a keystore with a self-signed certificate.
Go to .../your jre directory/jre/bin/

Keytool is a key and certificate management tool.
To create a new keystore with a self-signed certificate run this command:

```
keytool -genkey -alias myalias -keystore mykeystore -storepass passwd
```

Let's look at what each of the keytool sub-parts mean.

- The command for generating keys is *-genkey.*
- The *-alias myalias* sub-part indicates the alias to be used in the future to refer to the keystore entry containing the keys that will be generated.
- The *-keystore mykeystore* sub-part indicates the name (and optionally path) of the keystore you are creating or already using.
- The *-storepass passwd* sub-part indicates the keystore password. If you don't include a -storepass option, you will be prompted for the keystore password.

For security reasons you should not normally set your key or keystore passwords on the command line, because they can be intercepted more easily that way. Instead you should leave off the -keypass and the -storepass options and type your passwords when you are prompted for them.

After entering the preceding keystore command, you will be prompted for your distinguished-name information. Following prompts are as follows:

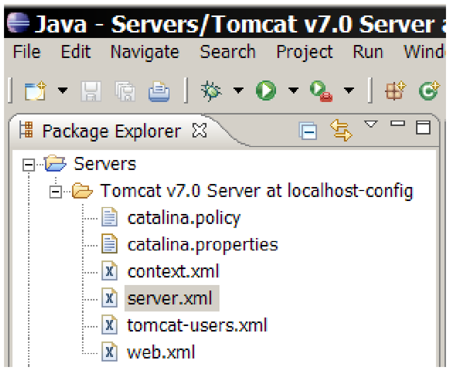
```
What is your first and last name?
  [Unknown]:  John Doe
What is the name of your organizational unit?
  [Unknown]:  Unknown
What is the name of your organization?
  [Unknown]:  ABC
What is the name of your City or Locality?
  [Unknown]:  New York
What is the name of your State or Province?
  [Unknown]:  NY
What is the two-letter country code for this unit?
  [Unknown]:  US
Is <CN=John Doe, OU=Unknown, O=ABC,
    L=New York, ST=NY, C=US> correct?
  [no]:  y
```

The result of using the keytool command is the creation of a keystore named mykeystore (if it doesn't already exist) in the same directory in which the command is executed with an assigned password mypasswd. The command generates a public/private key pair for the entity whose details have been entered while responding to prompts.

The command creates a self-signed certificate that includes the public key and the distinguished-name information. (The distinguished name you supply will be used as the "subject" field in the certificate.) This certificate will be valid for 90 days, the default validity period if you don't specify a *-validity* option.

When the keystore has been created, you can proceed with the reconfiguration of Tomcat.

In the package explorer go to Servers → Tomcat Server → server.xml

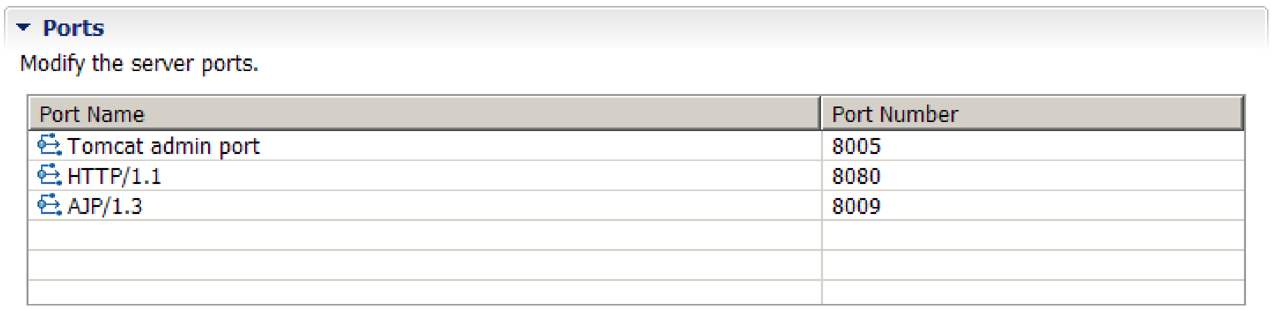

Java - Servers/Tomcat v7.0 Server
File Edit Navigate Search Project Run Wind
Package Explorer
Servers
Tomcat v7.0 Server at localhost-config
catalina.policy
catalina.properties
context.xml
server.xml
tomcat-users.xml
web.xml

You need to define a SSL connector, supplying it with the path and password to the keystore that has just been created.

```
 <Connector connectionTimeout="20000" port="8080" protocol="HTTP/1.1"
redirectPort="8443"/>
    <!-- A "Connector" using the shared thread pool-->
    <!--
    <Connector executor="tomcatThreadPool"
               port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000"
               redirectPort="8443" />
    -->
    <!-- Define a SSL HTTP/1.1 Connector on port 8443
         This connector uses the JSSE configuration, when using APR, the
         connector should be using the OpenSSL style configuration
         described in the APR documentation -->

<Connector
SSLEnabled="true"
clientAuth="false"
keystoreFile="C:\Documents and Settings\Administrator\Desktop\server.keystore"
keystorePass="keystorepass" m
axThreads="200"
port="8443"
protocol="org.apache.coyote.http11.Http11Protocol"
scheme="https"
secure="true"
sslProtocol="TLS"/>
```

Notice the change in server ports:

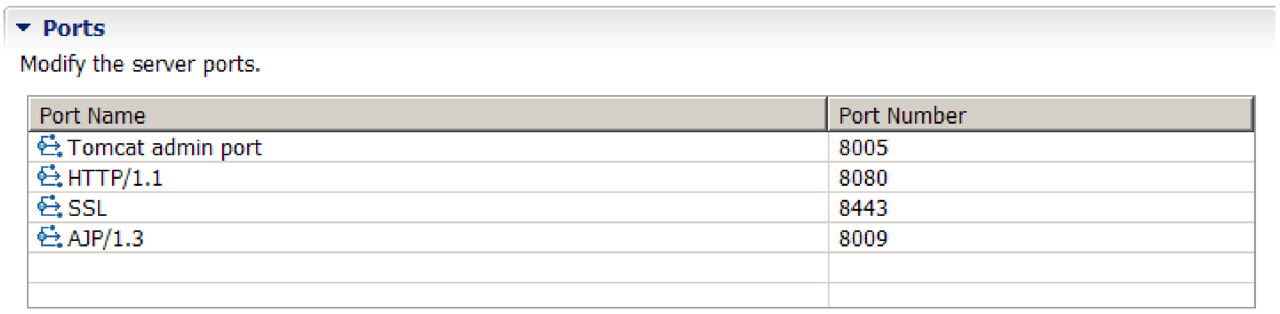
Before:


Ports

Modify the server ports.

Port Name	Port Number
Tomcat admin port	8005
HTTP/1.1	8080
AJP/1.3	8009

After:


Ports

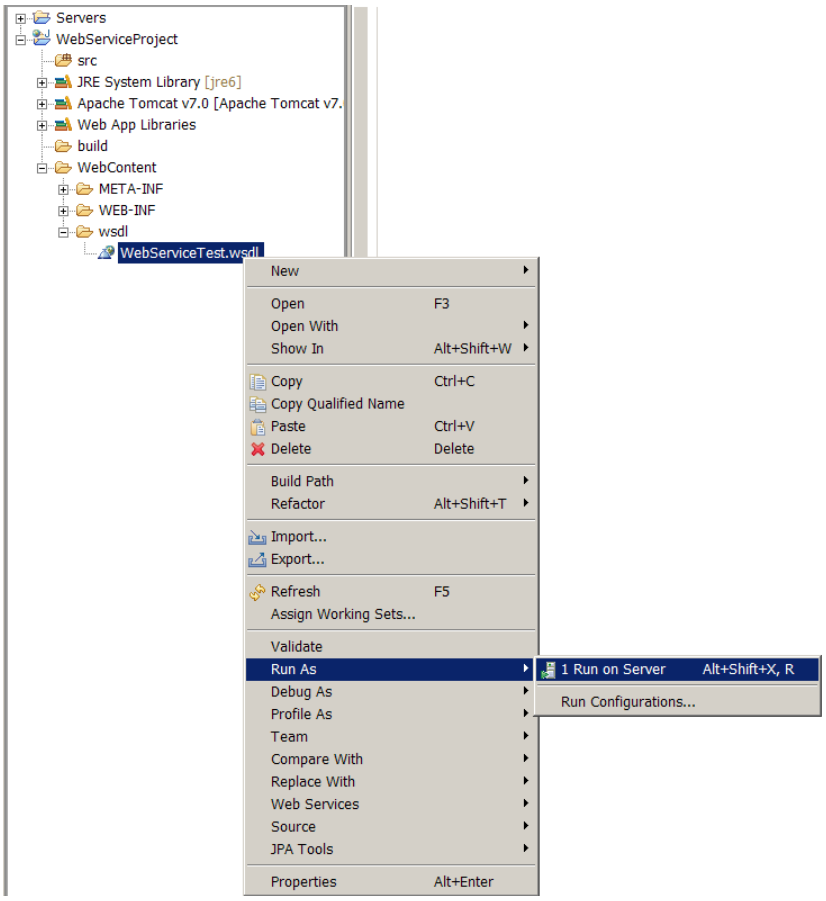
Modify the server ports.

Port Name	Port Number
Tomcat admin port	8005
HTTP/1.1	8080
SSL	8443
AJP/1.3	8009

In the WSDL of the Web Service explicitly change the endpoint:

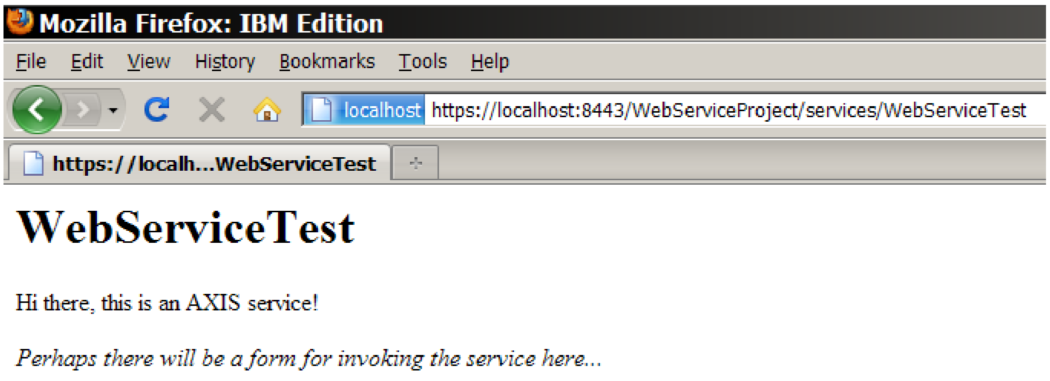
```
<wsdl:port binding="impl:WebServiceTestSoapBinding" name="WebServiceTest">
  <wsdlsoap:address
location="https://localhost:8443/WebServiceProject/services/WebServiceTest" />
  </wsdl:port>
```

Choose to run the WSDL on the server:



After restarting the server the WSDL should be available at:
https://localhost:8443/WebServiceProject/services/WebServiceTest?wsdl

Check through your web browser if the Web Service is running:


Mozilla Firefox: IBM Edition
File Edit View History Bookmarks Tools Help
localhost https://localhost:8443/WebServiceProject/services/WebServiceTest
https://localh...WebServiceTest

WebServiceTest

Hi there, this is an AXIS service!

Perhaps there will be a form for invoking the service here...

Testing the created Web Service that runs via SSL with SoapUI

If you have installed SoapUI as a plug-in in Eclipse, switch to the SoapUI perspective.

Create a new SoapUI project, specifying the WSDL of the Web Service – the WSDL of the above example is available at: https://localhost:8443/WebServiceProject/services/WebServiceTest?wsdl
Leave all other options at their default values.

Once the project has been created, in the SoapUI navigator expand the project tree until reaching the methods of the Web Service.

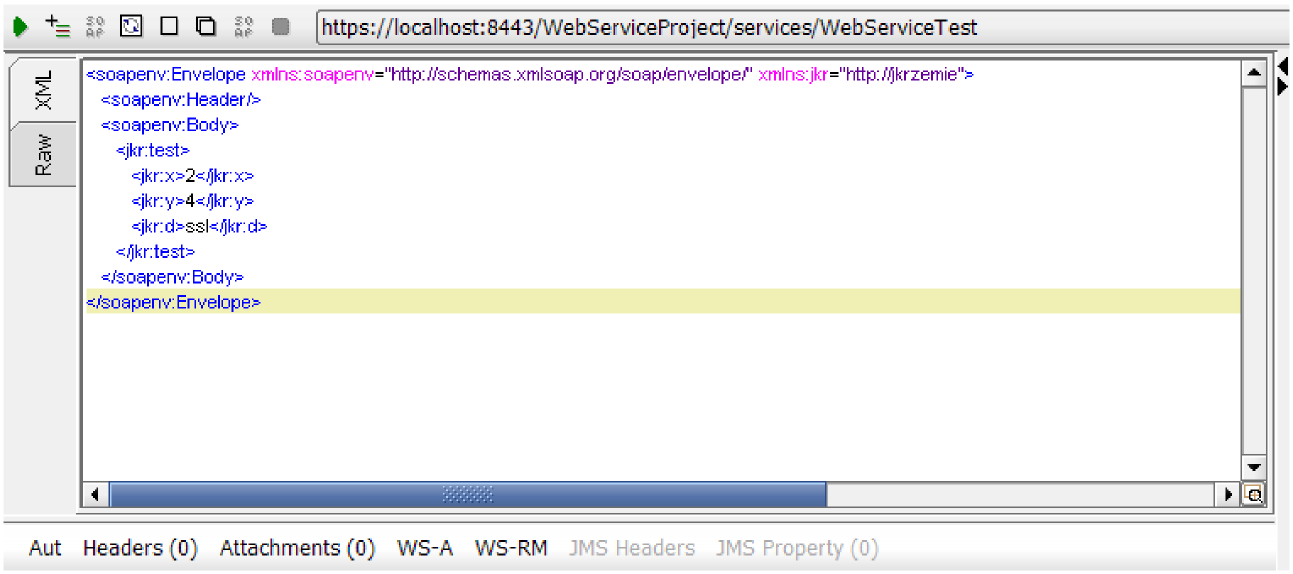
Double click on the request of the method you want to test. SoapUI should propose a SOAP request message.

Fill it in with appropriate values and click on the submit green arrow placed above the message window.

Check if the received response is as expected.

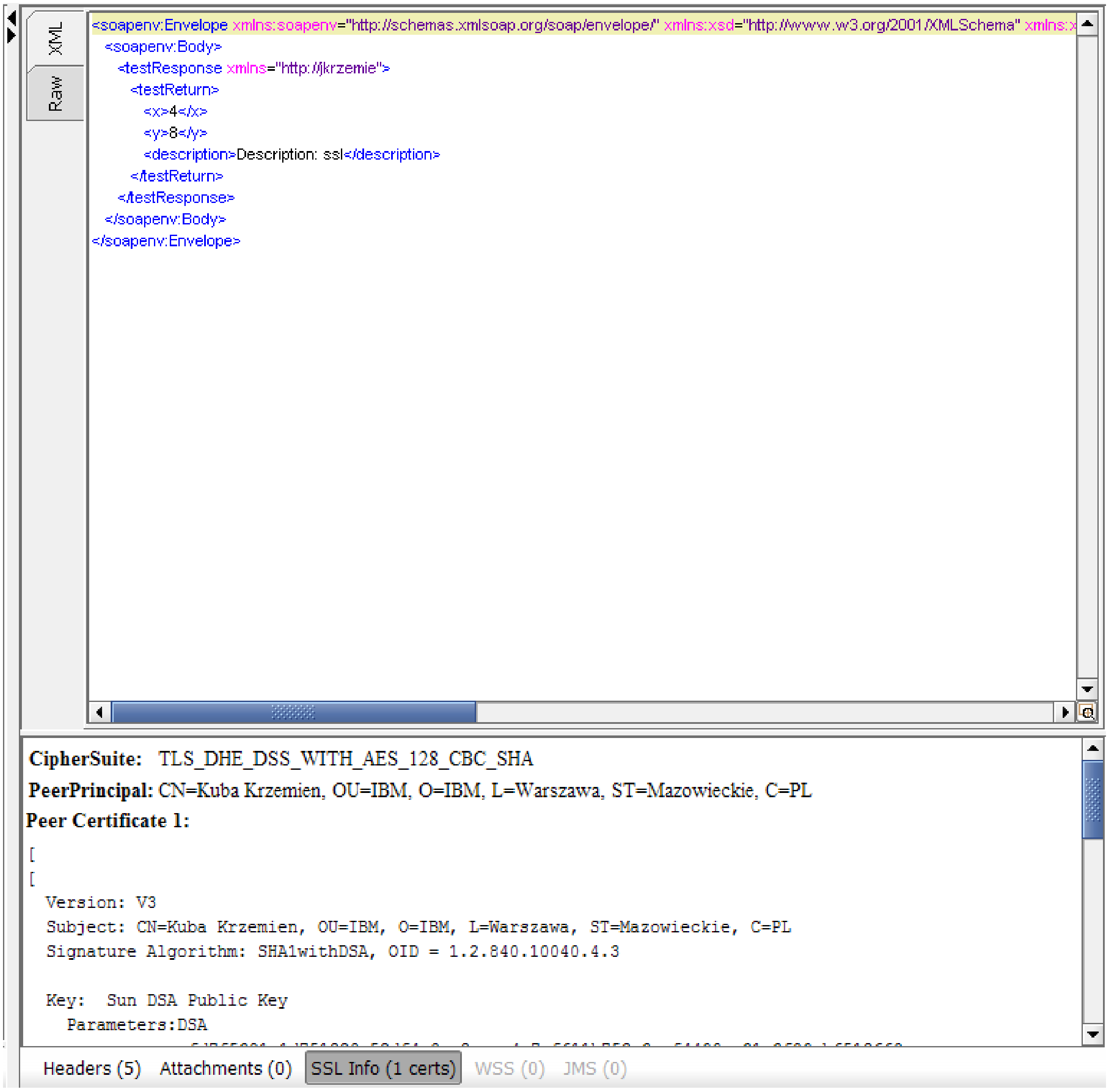
Notice the „SSL info” tab that is now available beneath the received response.

Request:



```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:jkr="http://jkrzemie">
  <soapenv:Header/>
  <soapenv:Body>
    <jkr:test>
      <jkr:x>2</jkr:x>
      <jkr:y>4</jkr:y>
      <jkr:d>ssl</jkr:d>
    </jkr:test>
  </soapenv:Body>
</soapenv:Envelope>
```

Response:



```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:x
  <soapenv:Body>
    <testResponse xmlns="http://jkrzemie">
      <testReturn>
        <x>4</x>
        <y>8</y>
        <description>Description: ssl</description>
      </testReturn>
    </testResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

CipherSuite: TLS_DHE_DSS_WITH_AES_128_CBC_SHA
PeerPrincipal: CN=Kuba Krzemien, OU=IBM, O=IBM, L=Warszawa, ST=Mazowieckie, C=PL
Peer Certificate 1:

```
[
[
  Version: V3
  Subject: CN=Kuba Krzemien, OU=IBM, O=IBM, L=Warszawa, ST=Mazowieckie, C=PL
  Signature Algorithm: SHA1withDSA, OID = 1.2.840.10040.4.3

  Key:  Sun DSA Public Key
    Parameters:DSA
```

The SSL Info Tab is one of the features of SoapUI that is not available in the standard Eclipse Web Service Explorer.

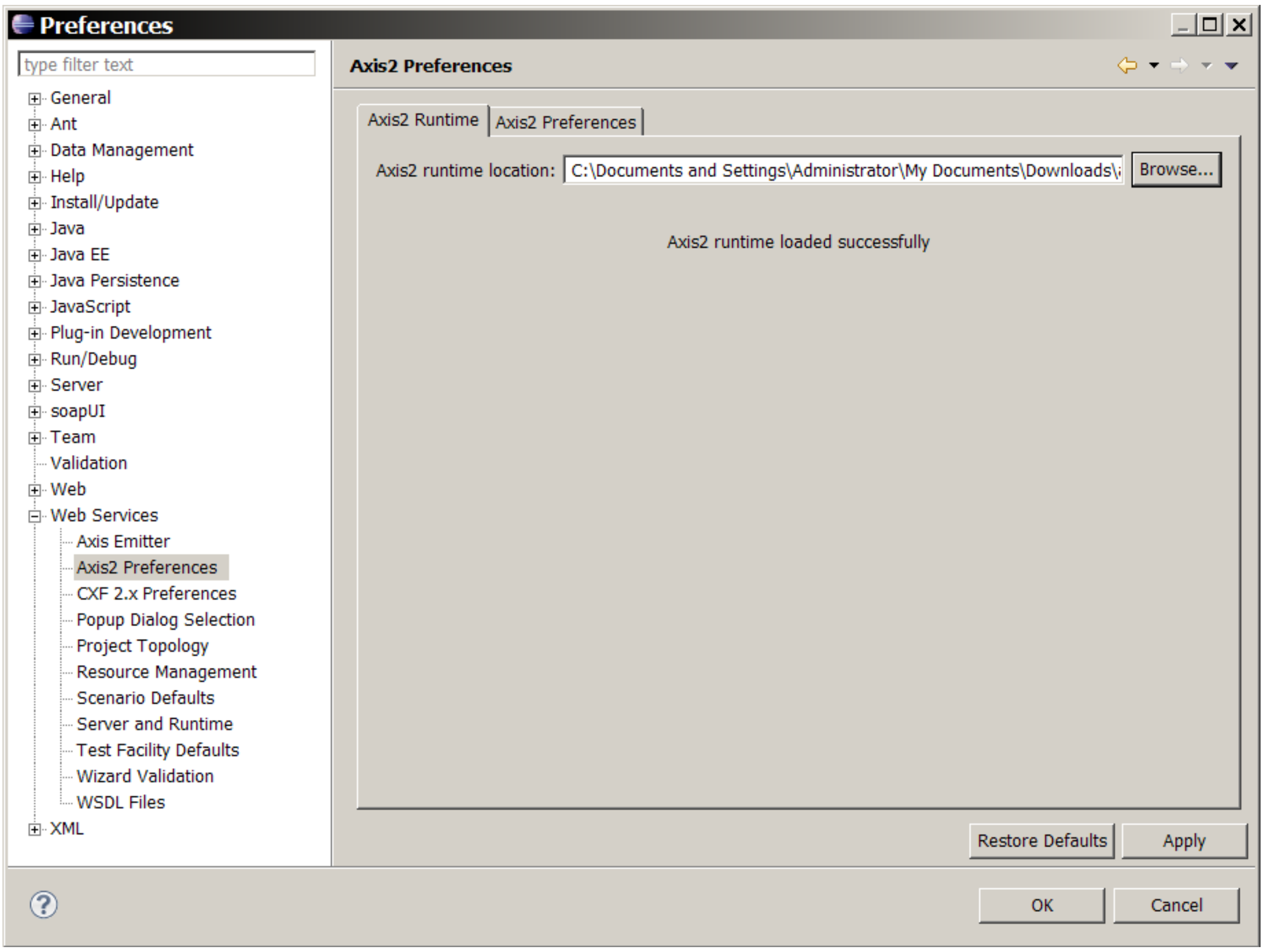
Creating a WS-Security enabled Web Service (Plain Text Auth)

Required tools:
- Axis2 (http://axis.apache.org/axis2/java/core/)
- Rampart (http://axis.apache.org/axis2/java/rampart/)
- WSS4J (http://ws.apache.org/wss4j/)

Apache Axis2 is a Web Service engine, Apache Rampart is the security module of Axis2.
Both are required to create a WS-Security enabled Web Service.
WSS4J is primarily a Java library that can be used to sign and verify SOAP Messages with WS-Security information. Rampart uses WSS4J, unfortunately as of the latest release, it does not include it in its distribution – you have to download it yourself.

Download the most recent binary version releases of both Axis2 and Rampart from the links provided above. Also download WSS4J, if not included in Ramparts distribution.
Depending on the version of the downloaded distribution, mex-1.5.2-impl.jar file can also be missing from the Axis2 library directory. If you can't find it or if you get a `java.lang.NoClassDefFoundError: org/apache/axis2/mex/MexException`, download the mex-1.5.2-impl.jar and add it to the /WebContent/WEB-INF/lib directory.

Next, in the upper bar of Eclipse choose Window → preferences → Web Services → Axis2 preferences and enter the correct Axis2 runtime location.



Axis2 runtime location: C:\Documents and Settings\Administrator\My Documents\Downloads\

Axis2 runtime loaded successfully

Now create a new Dynamic Web Project.
Change the dynamic web module version to 2.5.
Once the project is created, add to its buildpath the WSS4J library.

Add all Rampart libraries from the downloaded distribution to /WebContent/WEB-INF/lib.
Add all Rampart modules from the downloaded distribution to /WebContent/WEB-INF/modules.

Add an example source file to the new project:

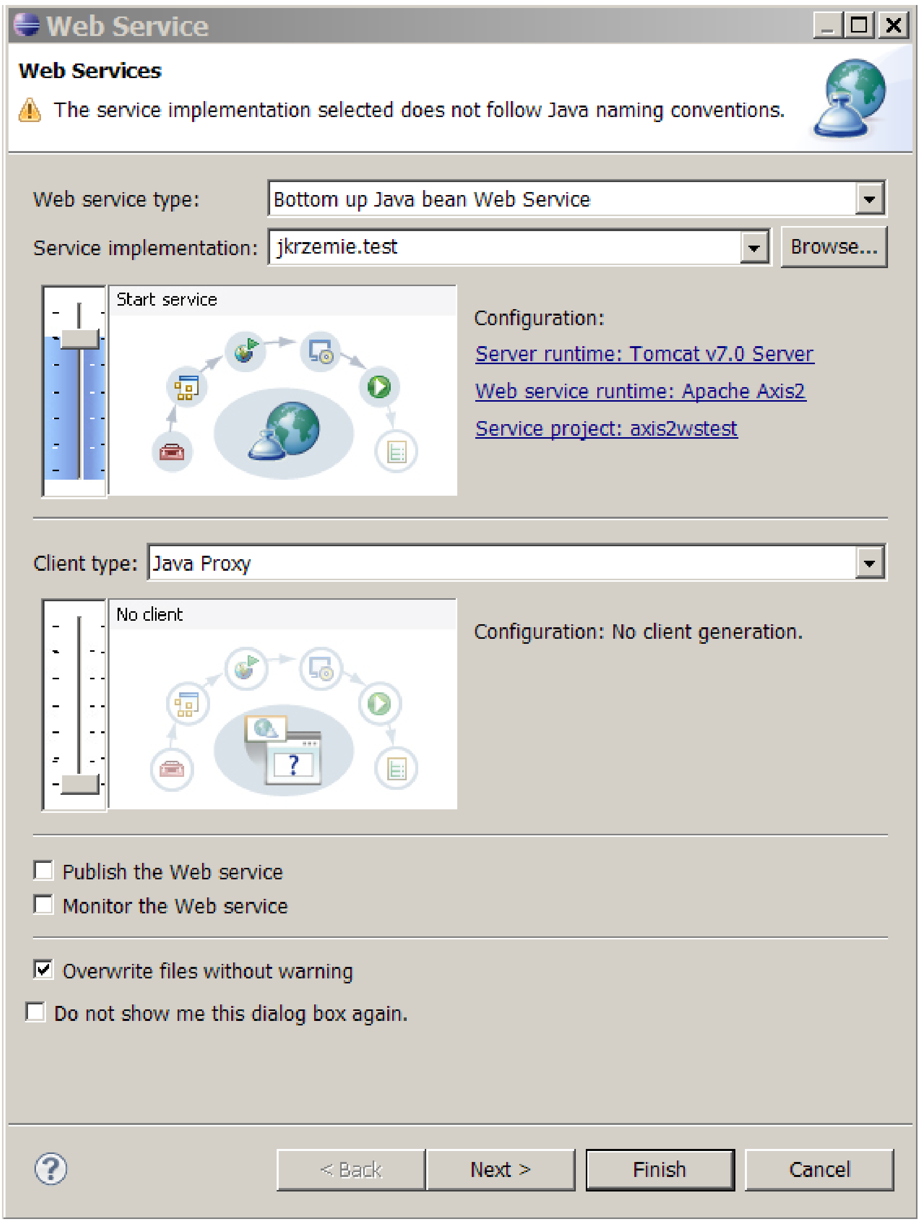
```
package axis2wstest;

public class test
{
      public int testws(int x)
      {
            return 2*x;
      }
}
```

Add a PWCBHandler class:

```
package axis2wstest;


import org.apache.ws.security.WSPasswordCallback;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;

import java.io.IOException;
public class PWCBHandler implements CallbackHandler {

    public void handle(Callback[] callbacks) throws IOException,
            UnsupportedCallbackException {

        for (int i = 0; i < callbacks.length; i++) {

            WSPasswordCallback pwcb = (WSPasswordCallback)callbacks[i];

            if(pwcb.getIdentifer().equals("test") &&
pwcb.getPassword().equals("pass")) {
                return;
            } else {
                throw new UnsupportedCallbackException(callbacks[i],
"Incorrect login/password");
            }

        }
    }

}
```

Create a Web Service from this source file.

Remember to select the correct Web Service runtime (Axis2).



Web Service

Web Services

⚠ The service implementation selected does not follow Java naming conventions.

Web service type: Bottom up Java bean Web Service

Service implementation: jkrzemie.test

Start service

Configuration:

Server runtime: Tomcat v7.0 Server

Web service runtime: Apache Axis2

Service project: axis2wstest

Client type: Java Proxy

No client

Configuration: No client generation.

☐ Publish the Web service

☐ Monitor the Web service

☑ Overwrite files without warning

☐ Do not show me this dialog box again.

Now in the package explorer open the file:
/WebContent/WEB-INF/services/test/META-INF/services.xml.

Add references to the Rampart module, configure Ramparts password callback class:

```
<service name="test">
<module ref="rampart"/>
      <Description>
            Please Type your service description here
      </Description>
      <messageReceivers>
            <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-only"
class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver" />
            <messageReceiver  mep="http://www.w3.org/2004/08/wsdl/in-out"
class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
      </messageReceivers>
      <parameter name="ServiceClass"
locked="false">axis2wstest.test</parameter>
    <operation name="testws">
        <messageReceiver
class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
    </operation>
<!-- Server policy for Username Token with plaintext password -->
<wsp:Policy wsu:Id="UsernameToken" xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
    xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
  <wsp:ExactlyOne>
    <wsp:All>
      <sp:TransportBinding
xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
         <wsp:Policy>
            <sp:TransportToken>
               <wsp:Policy>
                  <sp:HttpToken RequireClientCertificate="false"/>
               </wsp:Policy>
            </sp:TransportToken>
                        <sp:AlgorithmSuite>
                          <wsp:Policy>
                              <sp:Basic256/>
                          </wsp:Policy>
                        </sp:AlgorithmSuite>
         </wsp:Policy>
      </sp:TransportBinding>
      <sp:SupportingTokens
          xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702">
        <wsp:Policy>
          <sp:UsernameToken sp:IncludeToken="http://docs.oasis-open.org/ws-
sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRecipient"/>
        </wsp:Policy>
      </sp:SupportingTokens>

      <ramp:RampartConfig xmlns:ramp="http://ws.apache.org/rampart/policy">
        <ramp:passwordCallbackClass>axis2wstest.PWCBHandler</ramp:passwordCallb
ackClass>
      </ramp:RampartConfig>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>

</service>
```

After restarting the server, the WSDL for this Web Service should be available at:
http://localhost:8080/axis2wstest/services/test?wsdl

Testing the created WS-Security enabled Web Service with SoapUI

If you have installed SoapUI as a plug-in in Eclipse, switch to the SoapUI perspective.
Create a new SoapUI project, specifying the WSDL of the Web Service – the WSDL of the above example is available at: http://localhost:8080/axis2wstest/services/test?wsdl
Leave all other options at their default values.

Once the project has been created, in the SoapUI navigator expand the project tree until reaching the methods of the Web Service.

Double click on the request of the method you want to test. SoapUI should propose a SOAP request message.
Fill it in with appropriate values and click on the submit green arrow placed above the message window.

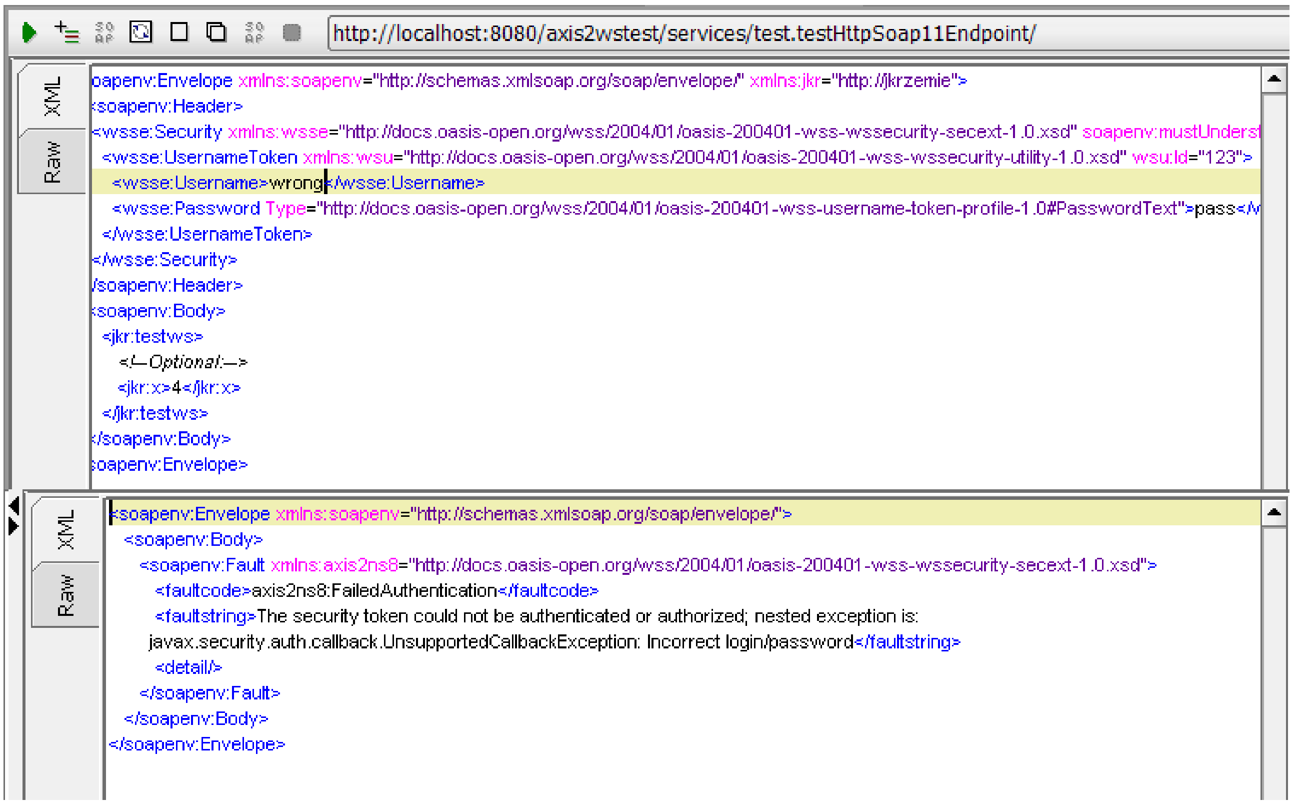
Example of a valid request message:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:jkr="http://jkrzemie">
   <soapenv:Header>
    <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-wssecurity-secext-1.0.xsd" soapenv:mustUnderstand="1">
      <wsse:UsernameToken xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" wsu:Id="123">
        <wsse:Username>test</wsse:Username>

        <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-username-token-profile-1.0#PasswordText">pass</wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </soapenv:Header>
   <soapenv:Body>
      <jkr:testws>
         <!--Optional:-->
         <jkr:x>4</jkr:x>
      </jkr:testws>
   </soapenv:Body>
</soapenv:Envelope>
```

The only difference from a standard SOAP request (as proposed by the SoapUI) is the addition of the Security header.
As seen in the example above, the Security header contains information about the username and password.

After sending the request notice the different response messages received, depending on the login/password information sent.

Incorrect login or password:

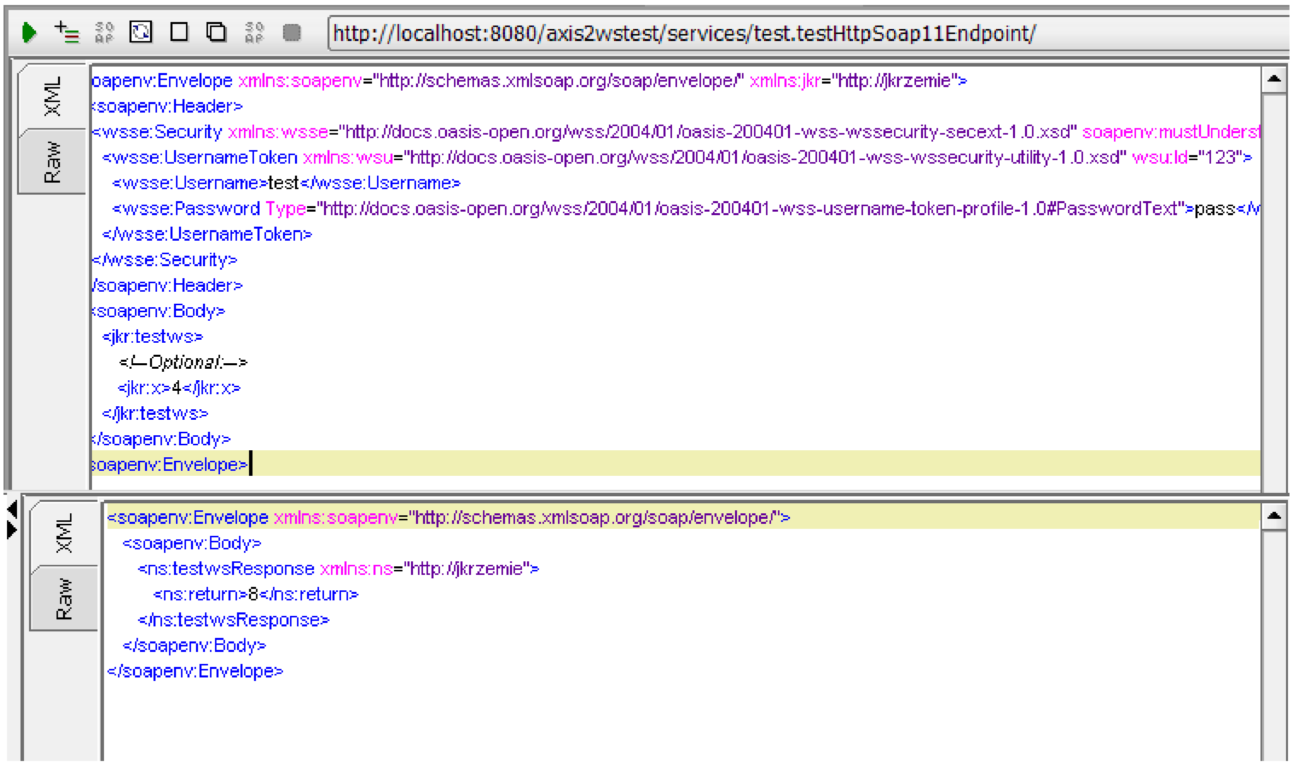


```
http://localhost:8080/axis2wstest/services/test.testHttpSoap11Endpoint/
```

```
oapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:jkr="http://jkrzemie">
<soapenv:Header>
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" soapenv:mustUnderst
  <wsse:UsernameToken xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" wsu:Id="123">
    <wsse:Username>wrong</wsse:Username>
    <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText">pass</w
  </wsse:UsernameToken>
</wsse:Security>
/soapenv:Header>
<soapenv:Body>
  <jkr:testws>
     <!--Optional:-->
     <jkr:x>4</jkr:x>
  </jkr:testws>
</soapenv:Body>
soapenv:Envelope>
```

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Body>
      <soapenv:Fault xmlns:axis2ns8="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
         <faultcode>axis2ns8:FailedAuthentication</faultcode>
         <faultstring>The security token could not be authenticated or authorized; nested exception is:
      javax.security.auth.callback.UnsupportedCallbackException: Incorrect login/password</faultstring>
         <detail/>
      </soapenv:Fault>
   </soapenv:Body>
</soapenv:Envelope>
```

Correct login and password:



```
http://localhost:8080/axis2wstest/services/test.testHttpSoap11Endpoint/
```

```
oapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:jkr="http://jkrzemie">
<soapenv:Header>
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" soapenv:mustUnderst
  <wsse:UsernameToken xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" wsu:Id="123">
    <wsse:Username>test</wsse:Username>
    <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText">pass</w
  </wsse:UsernameToken>
</wsse:Security>
/soapenv:Header>
<soapenv:Body>
  <jkr:testws>
     <!--Optional:-->
     <jkr:x>4</jkr:x>
  </jkr:testws>
</soapenv:Body>
soapenv:Envelope>
```

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Body>
      <ns:testwsResponse xmlns:ns="http://jkrzemie">
         <ns:return>8</ns:return>
      </ns:testwsResponse>
   </soapenv:Body>
</soapenv:Envelope>
```