
SilverStream eXtend Workbench

Tutorial: Developing Web Services

Version 4.0

Jun 2002

SilverStream[®]

Copyright ©2002 SilverStream Software, Inc. All rights reserved.

SilverStream software products are copyrighted and all rights are reserved by SilverStream Software, Inc.

SilverStream and jBroker are registered trademarks and SilverStream eXtend is a trademark of SilverStream Software, Inc.

Title to the Software and its documentation, and patents, copyrights and all other property rights applicable thereto, shall at all times remain solely and exclusively with SilverStream and its licensors, and you shall not take any action inconsistent with such title. The Software is protected by copyright laws and international treaty provisions. You shall not remove any copyright notices or other proprietary notices from the Software or its documentation, and you must reproduce such notices on all copies or extracts of the Software or its documentation. You do not acquire any rights of ownership in the Software.

Jakarta-Regexp Copyright ©1999 The Apache Software Foundation. All rights reserved. Ant Copyright ©1999 The Apache Software Foundation. All rights reserved. Xalan Copyright ©1999 The Apache Software Foundation. All rights reserved. Xerces Copyright ©1999-2000 The Apache Software Foundation. All rights reserved. Jakarta-Regexp, Ant, Xalan and Xerces software is licensed by The Apache Software Foundation and redistribution and use of Jakarta-Regexp, Ant, Xalan and Xerces in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notices, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear. 4. The names "The Jakarta Project", "Jakarta-Regexp", "Xerces", "Xalan", "Ant" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org <<mailto:apache@apache.org>>. 5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of The Apache Software Foundation. THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright ©1996-2000 Autonomy, Inc.

Copyright ©2000 Brett McLaughlin & Jason Hunter. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the disclaimer that follows these conditions in the documentation and/or other materials provided with the distribution. 3. The name "JDOM" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact license@jdom.org <<mailto:license@jdom.org>>. 4. Products derived from this software may not be called "JDOM", nor may "JDOM" appear in their name, without prior written permission from the JDOM Project Management (pm@jdom.org <<mailto:pm@jdom.org>>). THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Sun Microsystems, Inc. Sun, Sun Microsystems, the Sun Logo Sun, the Sun logo, Sun Microsystems, JavaBeans, Enterprise JavaBeans, JavaServer Pages, Java Naming and Directory Interface, JDK, JDBC, Java, HotJava, HotJava Views, Visual Java, Solaris, NEO, Joe, Netra, NFS, ONC, ONC+, OpenWindows, PC-NFS, SNM, SunNet Manager, Solaris sunburst design, Solstice, SunCore, SolarNet, SunWeb, Sun Workstation, The Network Is The Computer, ToolTalk, Ultra, Ultracomputing, Ultrasever, Where The Network Is Going, SunWorkShop, XView, Java WorkShop, the Java Coffee Cup logo, Visual Java, and NetBeans are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

IBM Jikes™ and Bean Scripting Framework (BSF) Copyright ©2001, International Business Machines Corporation and others. All Rights Reserved. This software contains code in executable form obtained pursuant to, and the use of which is subject to, the IBM Public License, a copy of which may be obtained at <http://oss.software.ibm.com/developerworks/opensource/license10.html>. Source code for Jikes™ is available at <<http://oss.software.ibm.com/developerworks/opensource/jikes/>>. Source code for BSF is available at <http://oss.software.ibm.com/developerworks/projects/bsf>.

SilverStream eXtend Workbench software contains Sun NetBeans software that has been modified by SilverStream. The source code for such software may be found at <http://www.silverstream.com/workbenchdownload> together with the Sun Public License that governs the use of such modified software. The Original Code is NetBeans. The Initial Developer of the Original Code is Sun Microsystems, Inc. Portions Copyright 1997-2000 Sun Microsystems, Inc. All Rights Reserved. The Contributor to Covered Code is SilverStream Software, Inc.

Graph Layout Toolkit and Graph Editor Toolkit (C) 1992 - 2001 Tom Sawyer Software, Oakland, California, All Rights Reserved.

This Software is derived in part from the SSLava™ Toolkit, which is Copyright ©1996-1998 by Phaos Technology Corporation. All Rights Reserved.

Contents

About This Book vii

Purpose	vii
Audience	vii
Prerequisites	vii
Organization	viii

Lesson 1 Registries and WSDL for Web Services 1

What you will learn	1
What you will do	1
Registries for Web Services	1
About registries	2
Browsing registries in Workbench	2
EXERCISE 1-1: Create a profile for a public registry	2
EXERCISE 1-2: Search for businesses	4
Information about businesses	5
Examining the information for a service	6
EXERCISE 1-3: Examine the services for a business	6
Information about services	8
Using the WSDL Editor	9
EXERCISE 1-4: Create a WSDL file for the Calculator Web Service	9
Tools for inserting elements	11
EXERCISE 1-5: Add a binding element	12
EXERCISE 1-6: Add a service element	13
Stylized view	15
EXERCISE 1-7: Change the Stylized view	16
WSDL Editor toolbar	17
EXERCISE 1-8: Generate a Java remote interface from WSDL	18
About publishing	21
Summary of what you've done	22

Lesson 2 Creating a Web Service 23

What you will learn	23
What you will do	23
Web Services using J2EE	24
JAX-RPC and RMI	24
Workbench and jBroker Web	24
Defining a WAR project for the service	25
EXERCISE 2-1: Set up directories for your project	25

EXERCISE 2-2: Create a new project	26
EXERCISE 2-3: Add source code to the project	28
EXERCISE 2-4: Add the jBroker Web libraries to the project	28
EXERCISE 2-5: Build the project	31
Generating Web Service code	31
EXERCISE 2-6: Run the Web Service Wizard	32
Getting ready to deploy	38
About the deployment descriptor	38
EXERCISE 2-7: Build the archive	38
EXERCISE 2-8: Examine the deployment descriptor	39
Deploying the project	41
EXERCISE 2-9: Deploy the project	41
Updating the J2EE server's classpath	44
Testing the Web Service	45
EXERCISE 2-10: Edit the test client code	45
EXERCISE 2-11: Test the Web Service with the generated client	46
Summary of what you've done	46
Lesson 3 Creating a Client Application for a Web Service	49
What you will learn	49
What you will do	49
Getting information about a Web Service	50
Setting up your project	50
EXERCISE 3-1: Set up a project directory and get the WSDL file	50
EXERCISE 3-2: Create a new project	51
EXERCISE 3-3: Set up a classpath for building the project	52
Generating client code from WSDL	53
EXERCISE 3-4: Generate client code from WSDL	53
Wizard results	56
Editing and testing the client application	56
EXERCISE 3-5: Edit the test client code	57
EXERCISE 3-6: Test the Web Service with the generated client	58
Summary of what you've done	59
Lesson 4 Using Web Services in a J2EE Web Application	61
What you will learn	61
What you will do	61
Defining a WAR project for the Web Service client application	62
EXERCISE 4-1: Create a new project	62
EXERCISE 4-2: Add the jBroker Web libraries to the project	64
Adding Web Service client code to the project	66
EXERCISE 4-3: Generate the client code for the Calculator Web	

Service	67
Creating a form that calls the Calculator Web Service	67
EXERCISE 4-4: Create a new JSP page	68
EXERCISE 4-5: Edit the JSP page	70
EXERCISE 4-6: Create a second JSP page to include in magicnumber.jsp	72
EXERCISE 4-7: Write a JavaBean to process the form	73
Deploying and testing the WAR	78
About the deployment descriptor	78
EXERCISE 4-8: Build the archive	78
EXERCISE 4-9: Edit the deployment descriptor	78
Deploying the project	81
EXERCISE 4-10: Deploy the project	81
EXERCISE 4-11: Test the Calculator Client application	84
Summary of what you've done	85
Lesson 5 Testing Techniques	87
What you will learn	87
What you will do	87
Viewing the WSDL in your browser	87
EXERCISE 5-1: View the WSDL for the deployed Web Service	88
Inspecting message traffic with TcpTunnel	89
EXERCISE 5-2: Edit the client code to redirect messages to TcpTunnel	89
EXERCISE 5-3: Run the client and observe the message traffic with TcpTunnel	90
Summary of what you've done	91

About This Book

Purpose

This tutorial shows you how to use SilverStream eXtend Workbench to develop a Web Service. You will learn about:

- Web Services and WSDL
- Registry Manager
- WSDL Editor
- Web Service Wizard
- Workbench projects
- Web applications packaged in J2EE WARs

Audience

This tutorial is for developers who want an introduction to Workbench projects while learning about building a Web Service.

Prerequisites

Experience This tutorial assumes you are a Java programmer who wants to use Workbench to develop J2EE applications. It assumes you have the following background:

- Experience with the Java programming language
- Understanding of the general structure of XML
- Understanding of a graphical development environment
- General understanding of J2EE concepts such as servlets
- Understanding of how browsers and application servers interact in Web applications

Software In addition to the Workbench software, you need:

- A J2EE application server for deploying the application

If you already have this software, you can deploy the standards-based J2EE WAR to your application server using Workbench deployment commands when available or your server's deployment tools.

If you don't have the required software, you can download the trial version of the SilverStream eXtend Application Server from www.silverstream.com/appserv-download.

Organization

Here's a summary of the lessons you'll find in this book:

Lesson		Description
1	Registries and WSDL for Web Services	Introduces the Registry Manager and the WSDL Editor
2	Creating a Web Service	Teaches how to use the Web Service Wizard to generate the files that wrap your Java class as a Web Service and how to deploy the Web Service as a WAR
3	Creating a Client Application for a Web Service	Teaches how to use the Web Service Wizard to generate files that a client application uses to call a remote Web Service
4	Using Web Services in a J2EE Web Application	Teaches how to build a Web application with a JSP page and JavaBean that call a Web Service; this client application uses the same code as was generated in the previous lesson
5	Testing Techniques	Demonstrates how a Web Service can return WSDL and how to use the TcpTunnel tool for viewing the SOAP messages sent between the client and the Web Service

1 Registries and WSDL for Web Services

What you will learn

This lesson describes the Workbench tools for working with online registries for Web Services. It also shows you how to use the WSDL Editor to create a file that describes a Web Service and can be published in a registry.

You will learn about:

- Registries for Web Services
- Browsing registries in Workbench
- Using the WSDL Editor

What you will do

1. Create a profile for a public registry
2. Search for businesses
3. Examine the services for a business
4. Create a WSDL file for the Calculator Web Service
5. Add a binding element
6. Add a service element
7. Change the Stylized view
8. Generate a Java remote interface from WSDL

How long will it take? About 20 minutes

NOTE You don't need to be running your J2EE application server for this lesson.

Registries for Web Services

When you want to make a Web Service publicly available or you want to find Web Services you can use, you use a registry. This lesson shows you how to identify a set of registries and how to search for offerings in those registries using the Registry Manager.

About registries

A Web Service registry is a repository of information about Web Services and other services. It supports finding and publishing information about a business and its services.

When providers create a Web Service, they can publish information about that service and their business in a registry so prospective consumers can discover the service and learn how to use it. When consumers want to find a Web Service, they can query the registry to find the services and businesses that fit their needs, and retrieve information about using those services.

Registries store this business and Web Service information in a standard XML-based format such as Universal Description, Discovery, and Integration (UDDI) or Electronic Business eXtensible Markup Language (ebXML). Typically businesses hosting registries provide Web page or GUI interfaces to publish to and query the registry. Other tools can use standard APIs to present their own interfaces.

Browsing registries in Workbench

The Registry Manager is on the Registries tab of the Navigation Pane. It displays registered businesses in the top panel and services in the lower panel. You can get listings from one or more registries.

To use the Registry Manager, you define profiles for the registries you want to access. (Profiles for several major registries are already defined for you.) Then you search for businesses or services by specifying a search string. Businesses or services that begin with that string are displayed in the browser.

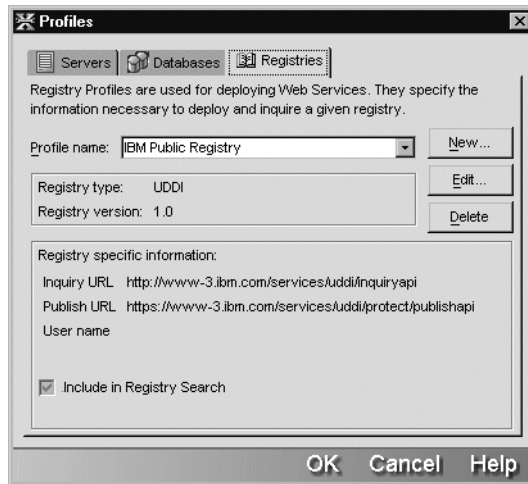


EXERCISE 1-1: Create a profile for a public registry

In this exercise you'll look at the profile for the IBM Public Registry that is defined for you when Workbench is installed.

1. Start Workbench. You can use the SilverStream eXtend Workbench shortcut on the Windows Start menu.
2. Select **Edit>Profiles** from the menu.

- In the Profiles dialog, select the **Registries** tab.



- Select **IBM Public Registry** in the **Profile name** dropdown list box.
- Click the **Edit** button to look at the profile. The profile has these values:

Option	Value
Profile name	IBM Public Registry
Registry type	UDDI
Inquiry URL	http://www-3.ibm.com/services/uddi/inquiryapi
Publish URL	https://www-3.ibm.com/services/uddi/protect/publishapi
User name Credential	Blank Later if you create an account with IBM, you can fill in your account information to enable publishing to the IBM registry.
Include in Registry Search	Selected You can prevent a registry from being searched by clearing this check box; you don't have to delete the profile

- Click **OK** to close the Edit a Registry Profile dialog.
- (Optional) Look at the profiles for the other registries that have been set up for you.

8. Click **OK** to close the Profiles dialog.



EXERCISE 1-2: Search for businesses

In this exercise you'll search for registered businesses whose names begin with X.

1. In the Navigation Pane, select the **Registries** tab.
The pane has two subpanes: Business and Service.
2. In the Business text box, type the letter **X** and click the curved blue arrow beside the text box.

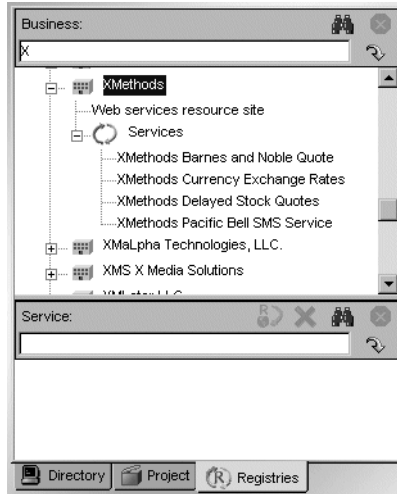
TIP You can enter multiple search terms by separating them with a vertical bar (|). For example, to find businesses that begin with X and W, type X|W.

The Registry Manager searches the registries you've defined. This can take a while, so be patient; the search can take from 15 seconds to 4 or 5 minutes. You can stop the search and look at partial results by clicking the red Stop button. When the button is no longer red, the search is done.

The results are displayed in the Business Pane. The first-level nodes in the expandable tree are registries, and the next level are business names.

3. Scroll through the list or collapse the top-level nodes to see which registries returned results. You can make the pane wider or longer for better viewing.
4. Expand the nodes for various businesses to see what information they provide.

- Find **XMethods**—it's in both registries—and expand its node to find out about its offerings.





Information about businesses

The information in a registry is self-supplied. A business tells you what they want you to know and selects their own categories.

The business section of a registry might include these types of information:

Information	Icon	Description
Business name		Business name used in this registry
Description	—	A short phrase describing the business
Categories		Categories to which the business belongs Classification schemes come from at least three sources: NAICS codes for industry segments, UNSPSC for product and service classifications, and geographic information

Information	Icon	Description
Identifiers		Information about the business, such as a DUNS number
Services		A list of services offered by the business, such as Web Services callable via HTTP and other services such as sales and technical support contact information You can select a service name to display its details in the lower pane

TIP Try other searches using the Advanced Search options (click the binoculars button). For more information about Advanced Search, see Registry Manager in the *Tools Guide*.

Examining the information for a service

Web Services are just a subset of the types of services that a business might publish in a registry. A business might list services such as sales and support contact information, as well as programmed Web Services.



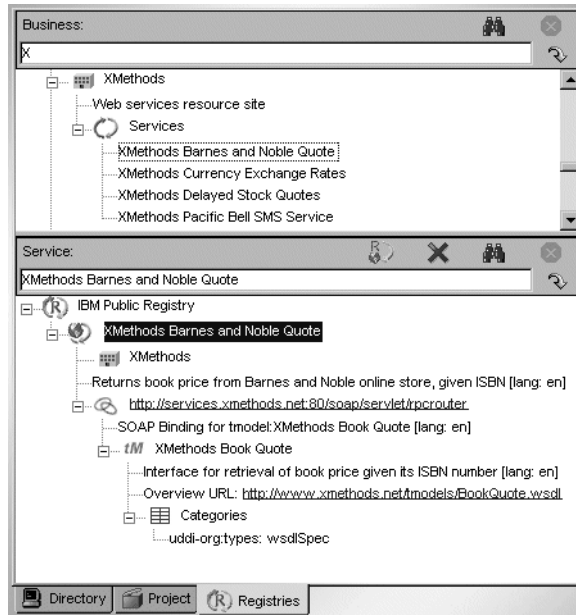
EXERCISE 1-3: Examine the services for a business

In this exercise you'll find out about the information available for a service and how to get a WSDL file for a service if one is available.

1. In the Business Pane of the Registry Manager, find the **XMethods** node and expand it to show its list of services.
2. Click the **XMethods Barnes and Noble Quote** service.

The Registry Manager retrieves information about the service and displays it in the Service Pane.

- Expand the nodes in the Service Pane to see all of the information. You can resize the pane so you can see more of the information at once.



- To retrieve the WSDL file that describes this service, highlight the line with the tModel icon—it says **XMethods Book Quote**.

 XMethods Book Quote

- Click the **Retrieve WSDL** icon in the toolbar above the Service text box.








The WSDL file for the Book Quote service opens in the WSDL Editor. For information about the WSDL Editor, see “Using the WSDL Editor” on page 9.

For the Book Quote service you could also click the Overview URL link to display the WSDL in your browser. A browser that understands XML is required.

Information about services

In the Service Pane, you can find out the technical details of a service offering. For a programmatically accessible service, the details include the URL for accessing the service and where to find information about the methods the service offers.

A service entry in a registry might include these types of information:

Information	Icon	Description
Service name		The name of the service
Business name		The business offering the service
Description	—	A short phrase describing the service
Binding		The URL for invoking the service
tModel		Data describing the service A UDDI registry stores the data as a tModel, which is a set of name/value pairs; the tModel node may be followed by a description
Overview URL	—	The URL of a document describing how to use the tModel data For a Web Service, this is usually a WSDL document.
Categories		Categories for the service The categorization has two parts: a name (for example, <code>uddi-org:types</code>) and a value (for example, <code>wsdlSpec</code>). The value <code>wsdlSpec</code> specifies that a WSDL document is available for the service. Other types of services can use other classification schemes.

TIP You can search for services without searching for businesses first. The basic search finds matches in service names, and Advanced Search (binoculars button) matches other services data. For more information, see Registry Manager in the *Tools Guide*.

Using the WSDL Editor

The WSDL Editor is an XML editor with extra features for handling WSDL elements. Most of the time, you will use WSDL definitions for Web Services that you get from registries or that you generate with the Web Service Wizard. However, if you need to edit a WSDL file, the editor comes in handy.

In Lesson 2, “Creating a Web Service”, you’ll build a Calculator Web Service. The Web Service Wizard generates WSDL to describe the service, so it’s not necessary to create one from scratch. But for this lesson, that’s what you’ll do.



EXERCISE 1-4: Create a WSDL file for the Calculator Web Service

In this exercise you’ll create a new WSDL file that describes the Calculator Web Service.

1. In Workbench, select **File>New** from the menu.
2. In the New File dialog, select the Web Services tab, highlight **WSDL**, and click **OK**.



3. In the WSDL Wizard, specify this information:

Option	Value
Definition Name	CalculatorService
Target Namespace	urn:CalculatorImpl
Documentation	The four basic arithmetic operations
Include WSDL template	Selected If you don't include the template, the wizard uses the definition name to name the file; but the other fields are ignored and the new file is empty

4. Click **Finish**.

Workbench starts the WSDL Editor and displays the beginning of a WSDL Web Service definition in the Edit Pane. If you compared this opening text with the file generated by the Web Service Wizard, you would see minor differences—but don't worry about it.



5. Select the following XML text and paste it into the editor on a blank line above the definitions end tag **</definitions>**. It's rather long because it defines request and response messages for all four arithmetic operations.

NOTE You could also use the editor's tools to insert the message and portType elements. These tools are described in the next section.

```
</types/>
<message name="subtractRequest">
  <part name="arg0" type="xsd:double"/>
  <part name="arg1" type="xsd:double"/>
</message>
<message name="subtractResponse">
  <part name="arg2" type="xsd:double"/>
</message>
<message name="divideRequest">
  <part name="arg3" type="xsd:double"/>
  <part name="arg4" type="xsd:double"/>
</message>
<message name="divideResponse">
  <part name="arg5" type="xsd:double"/>
</message>
<message name="DivideFault">
  <part name="reason" type="xsd:string"/>
  <part name="x" type="xsd:double"/>
  <part name="y" type="xsd:double"/>
</message>
<message name="addRequest">
  <part name="arg6" type="xsd:double"/>
  <part name="arg7" type="xsd:double"/>
```

```
</message>
<message name="addResponse">
  <part name="arg8" type="xsd:double"/>
</message>
<message name="multiplyRequest">
  <part name="arg9" type="xsd:double"/>
  <part name="arg10" type="xsd:double"/>
</message>
<message name="multiplyResponse">
  <part name="arg11" type="xsd:double"/>
</message>
<portType name="CalculatorImplWS">
  <operation name="subtract">
    <input message="tns:subtractRequest"/>
    <output message="tns:subtractResponse"/>
  </operation>
  <operation name="divide">
    <input message="tns:divideRequest"/>
    <output message="tns:divideResponse"/>
    <fault message="tns:DivideFault" name="fault1"/>
  </operation>
  <operation name="add">
    <input message="tns:addRequest"/>
    <output message="tns:addResponse"/>
  </operation>
  <operation name="multiply">
    <input message="tns:multiplyRequest"/>
    <output message="tns:multiplyResponse"/>
  </operation>
</portType>
```

Tools for inserting elements

The editor has dialogs to assist you with inserting top-level WSDL elements. The WSDL in the editor is missing two important elements: binding and service. You'll use the editor tools to add them.



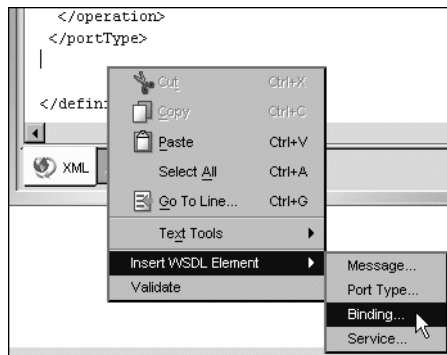
EXERCISE 1-5: Add a binding element

In this exercise you'll add a binding element, which specifies how messaging is handled.

1. In the Edit Pane near the end of the file, click to set the insertion point on a blank line between the end tags for portType and definitions.

```
</portType>
[insertion point here]
</definitions>
```

2. Right-click to display the WSDL popup menu, select **Insert WSDL Element**, then select **Binding** from the second menu.



3. In the Binding dialog, specify this information:

Option	Value
Name	CalculatorBinding
Documentation	SOAP Binding for Calculator service
Port Type	CalculatorImplWS TIP Use the dropdown list box to select a port type defined in the file
Binding Protocol	SOAP Binding Style: rpc Transport: http://schemas.xmlsoap.org/soap/http

The filled-in dialog looks like this:

4. Click **OK**.

The XML inserted in the file includes binding information for each operation defined in the portType element.



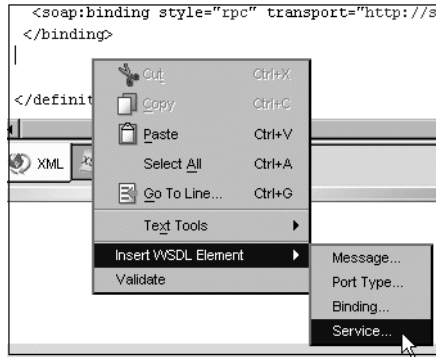
EXERCISE 1-6: Add a service element

In this exercise you'll add a service element, which specifies the URL a client application uses to invoke the deployed Web Service.

1. In the Edit Pane near the end of the file, click to set the insertion point on a blank line between the end tags for binding and definitions.

```
</binding>
[insertion point here]
</definitions>
```

2. Right-click to display the WSDL popup menu, select **Insert WSDL Element**, then select **Service** from the second menu.



3. In the Service dialog, specify this information:

Option	Value
Name	CalculatorService
Documentation	URL for locally deployed Calculator Web Service

4. Click **Add** to add a line for port information.
5. Enter these values to describe the port:

Option	Value
Name	CalculatorPort
Binding	CalculatorBinding TIP Use the dropdown list box to select a binding defined in the file
Address Type	SOAP TIP Use the dropdown list box to select a type
Location	http://localhost/ProverbsCloud/Calculator/CalculatorImpl NOTE Location is the URL where the Web Service will be deployed. For this lesson, use the sample URL above; you don't need a working URL yet.

The filled-in dialog looks like this:

Service

Enter information for the service element.

Name:
CalculatorService

Documentation:
URL for locally deployed Calculator Web Service

Ports:

Name	Binding	Address Type	Location
CalculatorPort	CalculatorBinding	SOAP	http://localhost/Prove

Add
Delete

OK Cancel Help

6. Click OK.

This XML is inserted in the file:

```
<service name="CalculatorService">
  <documentation>
    URL for locally deployed Calculator Web Service
  </documentation>
  <port name="CalculatorPort" binding="CalculatorBinding">
    <soap:address
      location="http://localhost/ProverbsCloud/Calculator/CalculatorImpl"/>
    </port>
</service>
```

The rest of this lesson shows you how to use some more features of the WSDL Editor.

Stylized view

The WSDL Editor has a second pane that displays the XML content of the WSDL document in a report format. You can customize the content and layout using XSL style sheets. You cannot edit in the Stylized view.



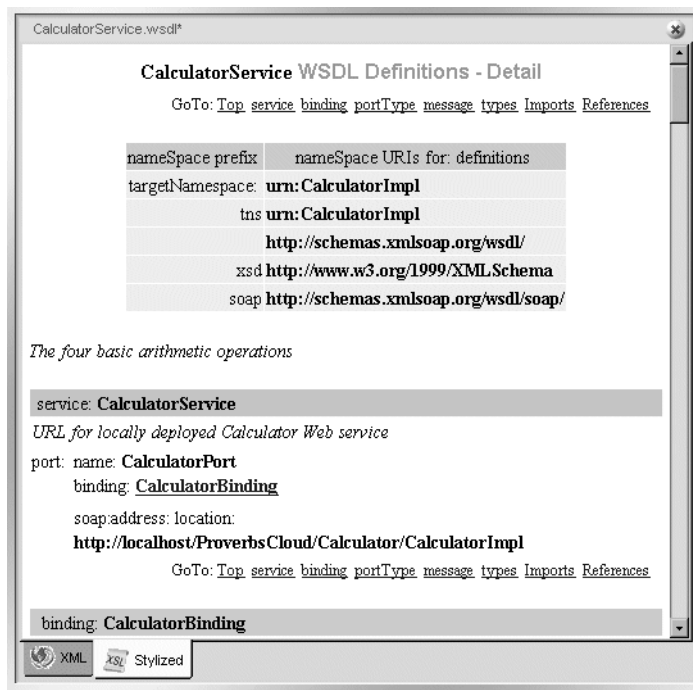
EXERCISE 1-7: Change the Stylized view

In this exercise you'll look at the views that are provided and find out where you can add your own custom view.

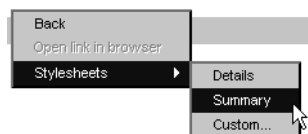
1. With the WSDL file open in the Edit Pane, click the **Stylized** tab at the bottom of the pane.



The format of the WSDL changes to the Details view, which presents the information in a more readable format.



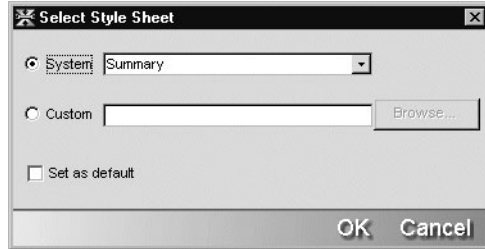
2. Right-click in the editing area, then select **Stylesheets** on the popup menu and **Summary** on the second menu.



Another formatted view appears.

3. Right-click again in the editing area, then select **Stylesheets** on the popup menu and **Custom** on the second menu.

In the Select Style Sheet dialog, you can choose the default style sheet: Details, Summary, or a custom style sheet. You can create your own XSL style sheets to present the information in different ways.



4. Click **Cancel** to close the dialog.
5. Click the **XML** tab to return to the editable view.
6. Save the file in a convenient directory, for example c:\WorkbenchProjects. Its name is **CalculatorService.wsdl**. Then close the file.

WSDL Editor toolbar

When you open the WSDL Editor, several buttons are added to the main toolbar.



You can:

- Validate the XML against the WSDL DTD
- Publish the WSDL to a UDDI registry defined in the Workbench registry profiles; you need an account with the registry you select
- Generate a Java class that matches the methods defined in the WSDL file

The next exercise will show you how to generate a Java class from WSDL.



EXERCISE 1-8: Generate a Java remote interface from WSDL

In this exercise you'll create a remote interface and other Web Service classes from a WSDL specification. You could use the resulting Java files to create a new Web Service or client application.

NOTE You need an open project for this exercise. You can use any project, since you won't be taking this any further than generating the code. Skip to Step 7 if you have an open project and want to generate the files there.

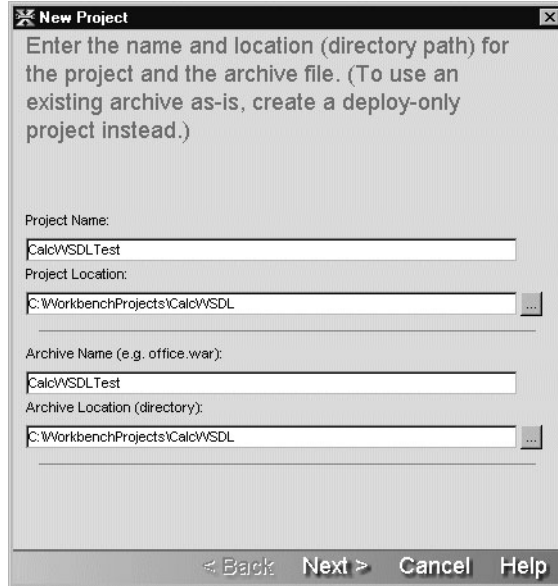
1. In Workbench, close any open projects and select **File>New Project** from the menu.
2. In the New Project Wizard, select **JAR** and then click **OK**.



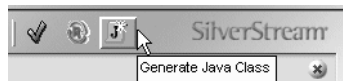
3. In the Project Name field, type **CalcWSDLTest**.
4. Click the ellipses beside the **Project Location** field and select a directory where you want to put Workbench projects, then type a new directory name (such as **CalcWSDL**). The Project Location field should end up with a value like this:

`C:\WorkbenchProjects\CalcWSDL`

The rest of the dialog is filled in automatically.

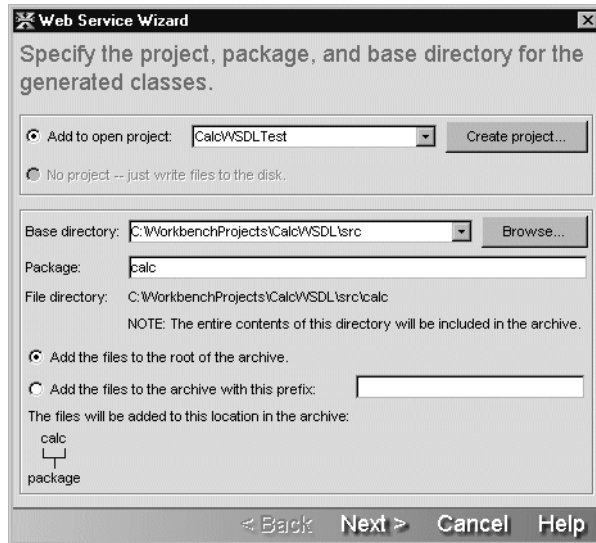


5. Click **Next**. If the project location directory doesn't exist, confirm that you want to create it.
6. On the last panel, check the project details then click **Finish**.
7. Select **File>Recent Files** and open **CalculatorService.wsdl** again.
8. In the WSDL toolbar, click the **Generate Java Class** button.



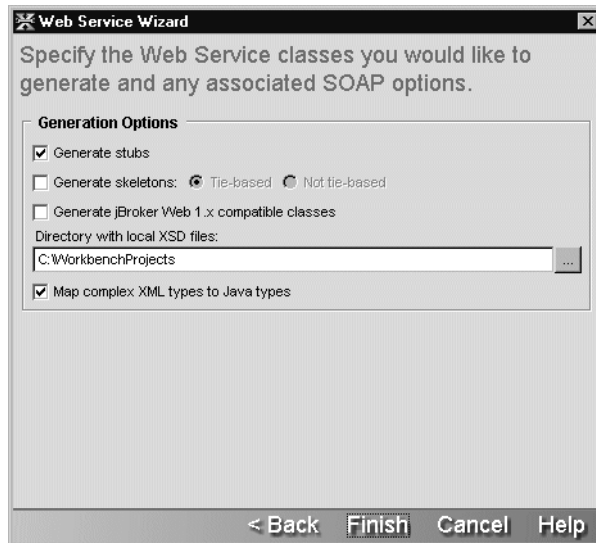
Workbench displays the project location panel of the Web Service Wizard.

9. Fill in the panel as shown below. The only value you should have to specify is the **calc** package.



Once you click **Next**, Workbench displays the class-generation and SOAP options panel of the Web Service Wizard.

10. Examine the settings on this panel (you don't need to change any of them).



These settings tell the wizard to generate stub classes for a Web Service client. It will put the generated files in the `src\calc` directory and add them to your project. You will learn more about these options in Lesson 2, “Creating a Web Service” and Lesson 3, “Creating a Client Application for a Web Service”.

11. Click Finish.

When the wizard finishes generating its output, you’ll find the generated files in the `src\calc` directory under the project root directory.

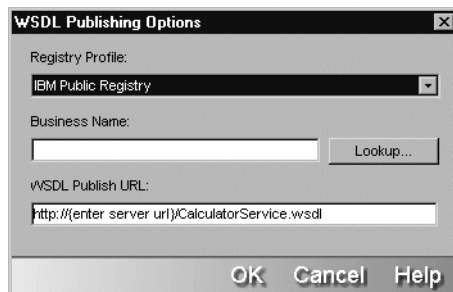
Typically you use a WSDL file as the starting point for generating stubs (including a remote interface and related classes) for a client application. However, if you generate skeletons, you’ll have all you need to begin building a Web Service. The other code you need is a class that implements the remote interface with the business logic for the Web Service methods.

About publishing

Workbench also provides facilities for publishing information about your service.

For most registries, you need to set up an account before you can publish. The registry profiles can store the URL for publishing and your ID and password.

TIP To publish, you open a WSDL file describing your service in the Edit Pane. The toolbar for the WSDL Editor includes a **Publish to Registry** button. It displays a dialog that lets you specify the registry, the business name, and the service URL. For more information, see Registry Manager in the *Tools Guide*.



Summary of what you've done

Using Workbench tools You used these tools in Workbench:

- Registry profiles (Edit>Profiles)
- Registry Manager (Registries tab of Navigation Pane)
- WSDL Editor

Next lesson In the next lesson you will learn about the Web Service Wizard. You'll create a project and develop the code for the Calculator Web Service.

2

Creating a Web Service

What you will learn

In this lesson you'll learn how to set up a WAR project for a Web Service and run the Web Service Wizard to generate SOAP processing code for the service. All you need to provide is the Java class that implements the methods that users of your service will call—in this sample, it will be a Calculator with these simple methods: add, subtract, multiply, and divide. Then you'll deploy the Calculator Web Service and test it with test tools provided by the wizard.

You will learn about:

- Web Services using J2EE
- Defining a WAR project for the service
- Generating Web Service code
- Getting ready to deploy
- Updating the J2EE server's classpath
- Testing the Web Service

What you will do

1. Set up directories for your project
2. Create a new project
3. Add source code to the project
4. Add the jBroker Web libraries to the project
5. Build the project
6. Run the Web Service Wizard
7. Build the archive
8. Examine the deployment descriptor
9. Deploy the project
10. Edit the test client code
11. Test the Web Service with the generated client

How long will it take? About 15 minutes

NOTE You do need to run your J2EE application server to deploy the Web Service you create in this lesson.

Web Services using J2EE

A Web Service is a component available on a remote server. Its interface is known, and you can call its methods via a standardized messaging protocol.

In the J2EE world, you make a Web Service available by deploying it as a servlet in a Web archive (WAR) on a J2EE application server. A client application makes a remote method call using SOAP XML messages. The SOAP dispatcher on the remote server receives the messages and directs the method call to the Web Service servlet. The Web Service wraps the return value as a SOAP message and sends it back to the client.

JAX-RPC and RMI

SilverStream supports the J2EE model for developing Web Services, which is based on JAX-RPC (Java API for XML-based RPC) and RMI (Java Remote Method Invocation). The business method signatures are declared in a remote interface. The service uses a skeleton class and the client uses a stub to manage the communication between the service and the client application.

Skeleton and tie The Web Service's skeleton class implements the remote interface. The skeleton receives a SOAP request, translates arguments from XML to Java data types, and calls the business method. The Web Service can also include a tie class that extends the skeleton and delegates the method call to another class that implements the business method.

Stub The client application uses a stub class that also implements the remote interface. When the client calls a method defined in the remote interface, the stub directs the call to the Web Service using an URL it has stored and transmits the method call as a SOAP message.

You don't need to worry about the implementation details of these classes; SilverStream provides tools that generate this code.

Workbench and jBroker Web

Workbench SilverStream eXtend Workbench provides a Web Service Wizard that generates the code for the communication between the Web Service and the client application. For the Web Service, all you need to provide is code for the business methods. For the client that calls a Web Service, you can generate the code from a WSDL file, which is an XML description of a Web Service.

jBroker Web jBroker Web is a JAX-RPC implementation that provides compilers and runtime support for Web Services on a J2EE application server. It's included in Workbench, and the Web Service Wizard uses its compilers to generate Web Service code. As you'll see, your deployed applications require access to jBroker Web and related API JARs.

jBroker Web includes command-line tools that invoke its compilers directly, but you will not use them in this tutorial. For more information, see the jBroker Web help.

The rest of this lesson teaches you how to build a Web Service, leading you through project setup, generating code with the wizard, and deploying and testing the result.

Defining a WAR project for the service

The Web Service Wizard in Workbench starts with a source object that implements or defines the business methods that you want to make available. There are several possible starting points in this process. You might begin with:

- A Java class that implements your business methods
- An interface that specifies signatures for your business methods
- An EJB session bean
- A WSDL service definition that specifies the operations of the Web Service

The Calculator Web Service uses the **CalculatorImpl** class, which defines methods for basic arithmetic. It also uses the **DivideFault** class, which handles divide-by-zero exceptions.

In this section you'll create a WAR project for the Calculator Web Service. First you'll do a little directory setup. Then you'll start Workbench to create the project file and add CalculatorImpl.java to the project.



EXERCISE 2-1: Set up directories for your project

In this exercise you will create directories for your source files.

1. Using your operating system tools, create a root directory for your project called **CalculatorWS**. You can put it at the root level of your disk drive or in a subdirectory of your choosing. The sample paths in the tutorial assume that you create CalculatorWS in the WorkbenchProjects directory. On Windows, it would look like this:

```
c:\WorkbenchProjects\CalculatorWS
```

2. In the CalculatorWS directory, create a subdirectory called **src**, and in the src directory, create a package subdirectory called **calc**.

3. Copy the files **CalculatorImpl.java** and **DivideFault.java** from the *Workbench-install-dir\docs\tutorial\TutorialFiles\webservices* directory to the **CalculatorWS\src\calc** project directory.

You now have a directory structure like this:

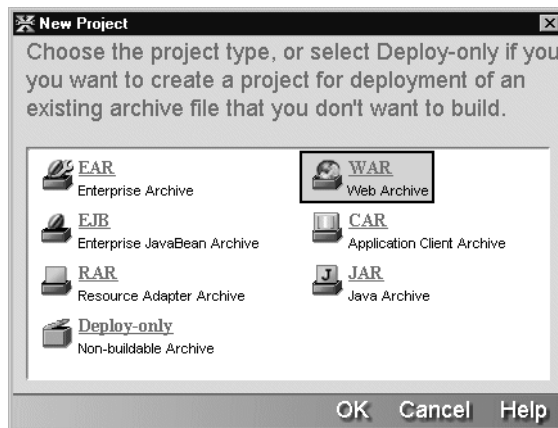
```
WorkbenchProjects\CalculatorWS\src\calc
```



EXERCISE 2-2: Create a new project

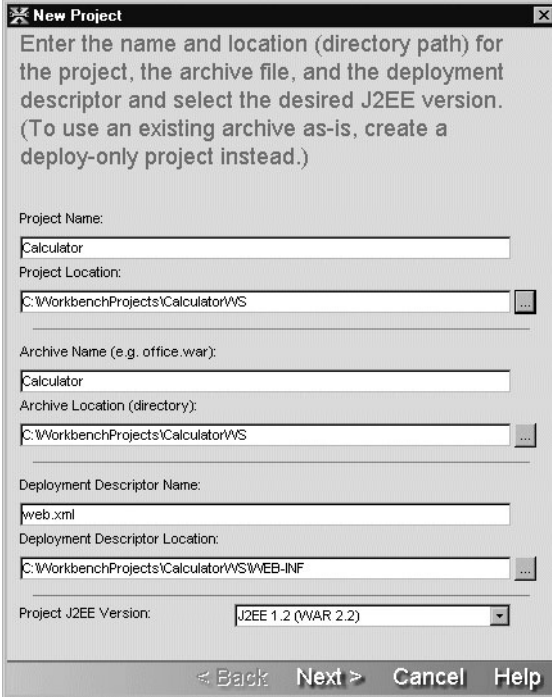
In this exercise you will start Workbench and use the New Project Wizard to create a project for the Calculator Web Service.

1. Start Workbench. You can use the SilverStream eXtend Workbench shortcut on the Windows Start menu.
OR
If Workbench is already running and a project is open, select **File>Close Project** from the menu. If prompted to close open files, click **Yes**.
2. Select **File>New Project** from the menu.
3. In the New Project Wizard, select **WAR** and then click **OK**.



4. In the Project Name field, type **Calculator**.
5. Click the ellipses beside the **Project Location** field and select the **CalculatorWS** directory you created in EXERCISE 2-1: “Set up directories for your project”. When you click **OK**, the rest of the dialog is filled in automatically.

- In the Project J2EE Version field, specify **J2EE 1.2 (WAR 2.2)** so your application will run on any server that supports J2EE 1.2 or 1.3.



New Project

Enter the name and location (directory path) for the project, the archive file, and the deployment descriptor and select the desired J2EE version. (To use an existing archive as-is, create a deploy-only project instead.)

Project Name:
Calculator

Project Location:
C:\WorkbenchProjects\Calculator\WS

Archive Name (e.g. office.war):
Calculator

Archive Location (directory):
C:\WorkbenchProjects\Calculator\WS

Deployment Descriptor Name:
web.xml

Deployment Descriptor Location:
C:\WorkbenchProjects\Calculator\WS\WEB-INF

Project J2EE Version:
J2EE 1.2 (WAR 2.2)

< Back Next > Cancel Help

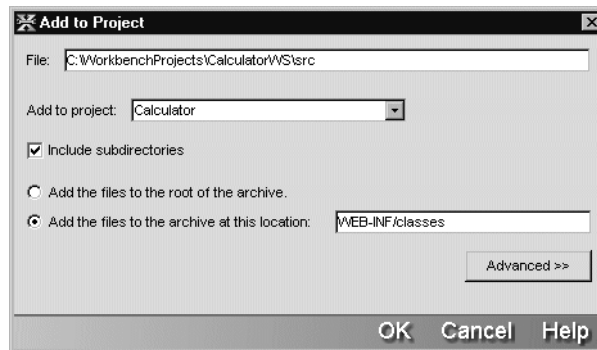
- Click **Next**.
- When the wizard asks if it should create the WEB-INF directory, click **Yes**.
The wizard summarizes the project information.
NOTE If another project was open when you selected New Project, you might see a panel about adding the project to the current project. If this happens, do not check the **Add this project** option. Click **Next** to go to the summary panel.
- Click **Finish**.
In the Navigation Pane, the Project tab displays the new project. You can use either a Source Layout view or an Archive Layout view.



EXERCISE 2-3: Add source code to the project

In this exercise you will add the src directory to the project and specify where it will be in the J2EE archive.

1. In the Navigation Pane, click the **Directory** tab.
2. Navigate to the **WorkbenchProjects/CalculatorWS/src** directory.
3. Right-click the **src** directory and select **Add to Project**.
4. In the Add to Project dialog, select **Add the files to the archive at this location**. In the text box, type **WEB-INF/classes**. Leave **Include subdirectories** selected.



5. Click **OK**.



EXERCISE 2-4: Add the jBroker Web libraries to the project

The Web Service uses classes in jbroker-web.jar and supporting JARs for SOAP message processing. In this exercise you will add these JARs to the archive for runtime access and to the project classpath for compile-time access.

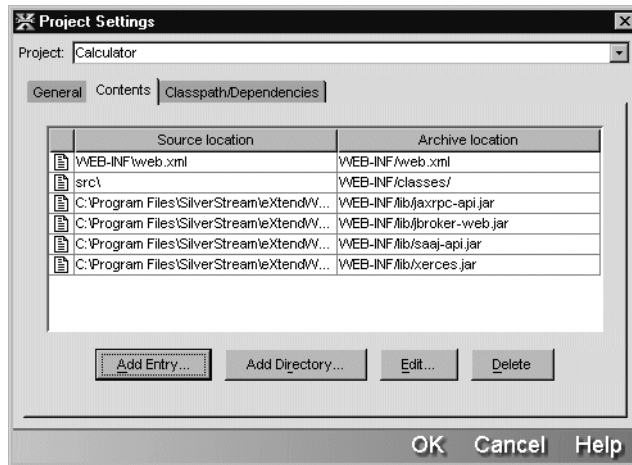
1. In Workbench, select **Project>Project Settings** from the menu.
2. Select the **Contents** tab and click the **Add Entry** button.
3. In the Select Contents dialog, navigate to the directory **Workbench-install-directory/compilelib**, then highlight the following files and click **Open**:
 - jaxrpc-api.jar
 - jbroker-web.jar
 - saaj-api.jar

- xerces.jar

The Add to Project dialog will prompt you for information about each file, one at a time.

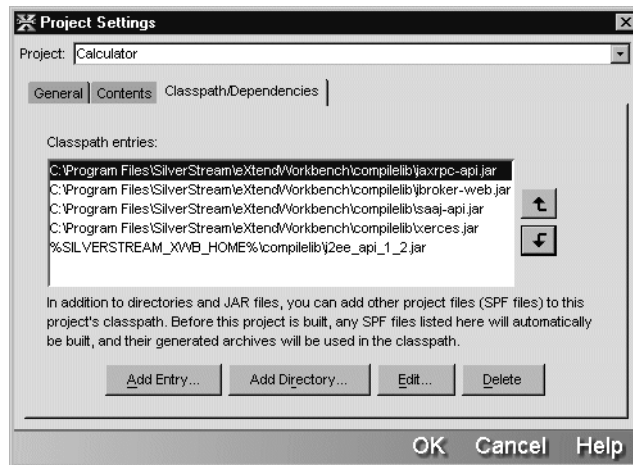
4. When you're prompted about jaxrpc-api.jar, select **Add the file to the archive at this location**. In the text box, type **WEB-INF/lib/jaxrpc-api.jar**. Then click **OK**.
5. When you're prompted about jbroker-web.jar, select **Add the file to the archive at this location**. In the text box, type **WEB-INF/lib/jbroker-web.jar**. Then click **OK**.
6. When you're prompted about saaj-api.jar, select **Add the file to the archive at this location**. In the text box, type **WEB-INF/lib/saaj-api.jar**. Then click **OK**.
7. When you're prompted about xerces.jar, select **Add the file to the archive at this location**. In the text box, type **WEB-INF/lib/xerces.jar**. Then click **OK**.

The WEB-INF/lib directory of the archive will now include these JARs.



8. Select the **Classpath/Dependencies** tab and click the **Add Entry** button.
9. In the Add to Classpath dialog, find the directory *Workbench-install-directory/compilelib* again, then highlight the following files and click **Open** then **OK**.
 - jaxrpc-api.jar
 - jbroker-web.jar
 - saaj-api.jar
 - xerces.jar

The Classpath/Dependencies tab should look something like this:



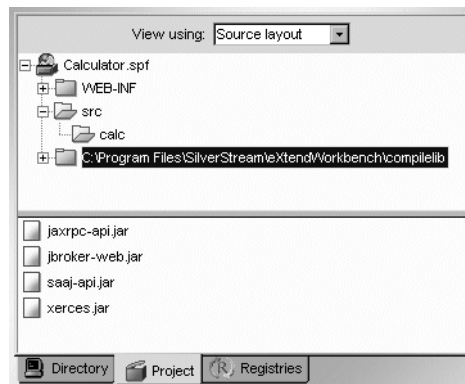
NOTE An archive of J2EE classes is already on the WAR's classpath. Its path uses an environment variable whose value is the Workbench install directory. The variable was defined when you installed Workbench.

10. Click **OK** to close the Project Settings dialog.

The project now includes references to the required JARs. When you build the archive, these JARs will be included. The JARs are also on the classpath for building the archive.

11. In the Navigation Pane, select the **Project** tab to see the project contents. Select **Source layout**, expand **src**, and select the **calc** directory. It contains CalculatorImpl.java and DivideFault.java. Click the **Workbench-install-directory\compilelib** entry to see jbroker-web.jar and the other JARs in the lower pane.

The expanded Source layout looks something like this:





EXERCISE 2-5: Build the project

The Web Service Wizard uses compiled files, so you need to build the project before invoking it.

- Select **Project>Build** from the menu.

If you get errors, the problem is probably in the classpath. Make sure you successfully completed EXERCISE 2-4: “Add the jBroker Web libraries to the project”.

Generating Web Service code

To convert your source object into a Web Service, you run the Web Service Wizard. It generates code that enables the server to translate XML SOAP requests into method calls for your source object.

TIP The Web Service Wizard requires that you have an open project. It puts the files it generates in that project.

About the URL for the Web Service When you run the wizard, one of the pieces of information you will provide is the URL that clients use to access the service. The URL has several parts:

Part	Description	Example
Server	URL for the server, including the port number (if not the default port 80) and any server-specific data TIP For a SilverStream server, include the database to which you deployed the WAR	http://localhost/Pr overbsCloud/ http://www.mydo main.com:8080/
Web application	URL for the WAR TIP For a SilverStream server, this is a relative URL that you specify in the deployment plan	Calculator/
Servlet mapping	URL for the servlet; this is the URL pattern assigned in the Servlet Mapping section of the deployment descriptor	CalculatorImpl

For example, when you deploy this Web Service to a local SilverStream server, the URL will be something like this:

```
http://localhost:80/ProverbsCloud/Calculator/CalculatorImpl
```



EXERCISE 2-6: Run the Web Service Wizard

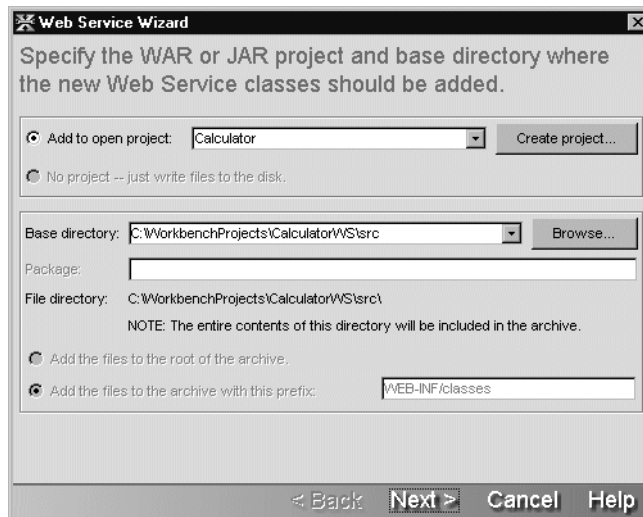
In this exercise you'll generate classes that turn CalculatorImpl into a Web Service.

1. With your project open in Workbench, select **File>New** from the menu.
2. In the New File dialog, select the **Web Services** tab, select **New Web Service**, and click **OK**.

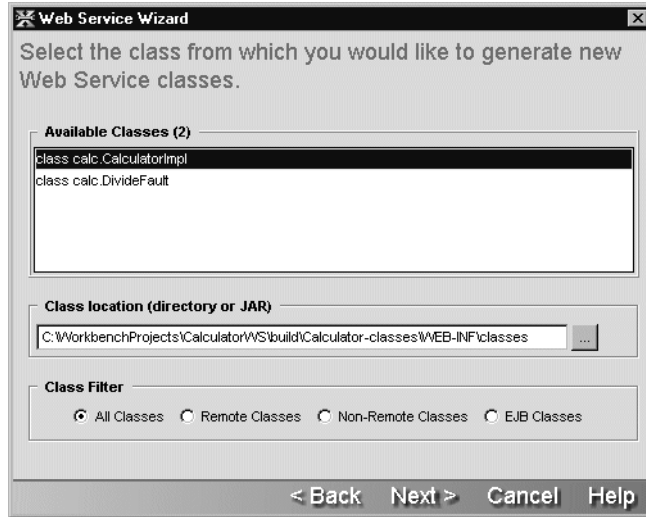


Workbench displays the project location panel of the Web Service Wizard.

3. You can accept the defaults on this panel (as shown below) and click **Next**.

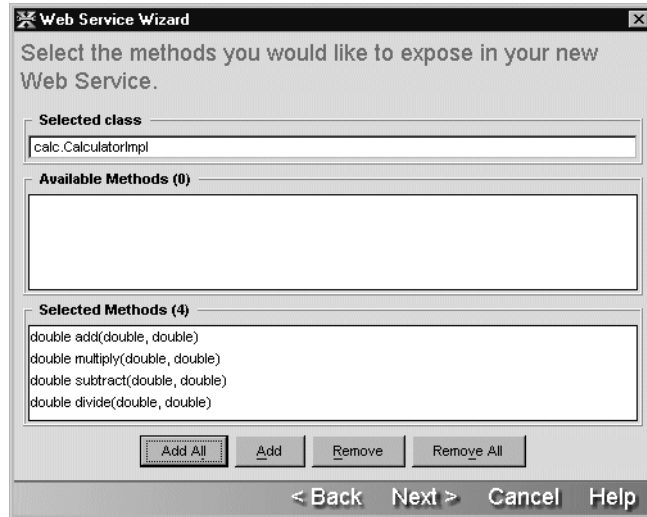


- When the class selection panel displays, highlight **class calc.CalculatorImpl** and click **Next**.



NOTE By default, the wizard displays the compiled classes of your project. You can optionally list classes located elsewhere (such as in an archive) and filter the list to show only specific kinds of classes.

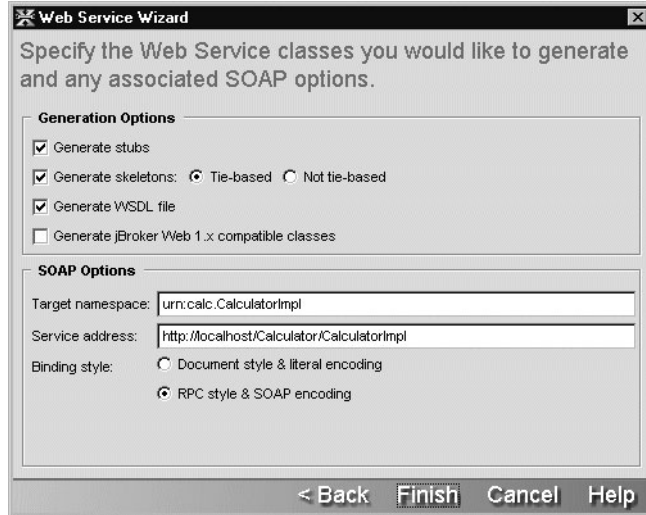
5. When the method selection panel displays, click **Add All** to use all four calculator methods. Then click **Next**.



The class generation and SOAP options panel displays.

6. In the **Service address** text box, specify the URL a client uses to access your service. The URL varies depending on your deployment server, as described in “About the URL for the Web Service” on page 31. For example, the URL for an application deployed to the ProverbsCloud database on a SilverStream server at www.mydomain.com might be:

`http://www.mydomain.com/ProverbsCloud/Calculator/CalculatorImpl`



7. Click **Finish**.

Wizard results

After you run the wizard, several files are added to the calc directory of your project. Because the wizard adds them to a project directory, they are automatically part of the project.

When you select all the generation options in the wizard, your project includes these files. The Java files are in the calc package directory.

File	Description	Where used
CalculatorImplWS.java	A remote interface that has declarations for the methods of your source object. It extends java.rmi.Remote. Each of the methods throws RemoteException.	Web Service and client program
CalculatorImplWS_ServiceSkeleton.java	A jBroker Web class that processes SOAP messages on the server. You should never need to modify this class.	Web Service

File	Description	Where used
CalculatorImplWS_ServiceTieSkeleton.java	A jBroker Web class that extends the ServiceSkeleton with a setTarget() method for identifying the object that implements the Web Service methods. You should never need to modify this class.	Web Service
CalculatorImplWSTie.java	A delegator class that extends the TieSkeleton and sets the tie's target to CalculatorImplWSDelegate.	Web Service
CalculatorImplWSDelegate.java	A delegator class that implements the remote interface and calls the methods of the source object. It implements all the constructors of the source object.	Web Service
CalculatorImplWSService.java	A Service interface used by JAX-RPC clients to obtain the stub for the Web Service. You should never need to modify this class.	Client program
CalculatorImplWSServiceImpl.java	A Service implementation class that handles instantiation of the stub (CalculatorImplWS_Stub). You should never need to modify this class.	Client program
CalculatorImplWS_Stub.java	A jBroker Web class that processes SOAP messages on the client. You should never need to modify this class.	Client program

File	Description	Where used
CalculatorImplWSClient.java	A standalone Java program for testing the Web Service. After you edit the code, use it to verify that the deployed Web Service works. The sample code is a model for code in a client program.	Testing only
CalculatorImplWS.wsdl	An XML description of the Web Service for publishing in a registry. This file is saved in the src directory of the project, not in the calc package directory.	Registry

The **delegator classes** CalculatorImplWSDelegate and CalculatorImplWSTie work together to bind the source object to the SOAP-processing objects. You don't have to edit anything to produce a working Web Service.

The **binding**—the URL for accessing the Web Service—is part of the code in the stub. You can override this URL in the client code that instantiates the stub.

NOTE If you run the wizard again, all these files get regenerated. Therefore if you need to change the code, it is better to define a class that extends the delegator class than to edit the generated code.

If the business logic isn't written If you had started this process with an interface instead of an implementation—if CalculatorImpl.java didn't exist and you had only the file CalculatorImplWS.java—you would need to write the business logic at this point. You could extend CalculatorImplWSTie or CalculatorImplWS_ServiceTieSkeleton and implement the business logic there, or write another class and set the target of CalculatorImplWSTie to point to it.

Getting ready to deploy

Workbench can build and deploy archives for any J2EE application server. These instructions provide the information you need to deploy this tutorial application. For details and server-specific information, see *Workbench Deployment Instructions*. (You can also use your own server tools to deploy.)

To deploy your Web Service, you will:

1. Build the archive
2. Look at servlet information that the wizard inserted in the deployment descriptor
3. Create a server profile (already done if you've deployed other applications to your server in Workbench)
4. Create a server-specific file with runtime deployment information
5. Specify Workbench deployment settings
6. Deploy to your server

About the deployment descriptor

When you created the project, Workbench created an XML descriptor file appropriate to the type of archive you selected. For a WAR, the file is called **web.xml**.

When you open **web.xml** for editing, the Deployment Descriptor Editor shows the XML elements in an expandable tree structure. You can also look at the raw XML. The editor uses the project's compiled code to determine what to show, which is why you build the archive first. If it isn't already built, Workbench offers to build it for you.

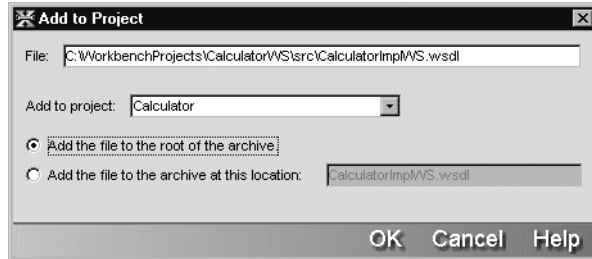


EXERCISE 2-7: Build the archive

In this exercise you'll include the generated WSDL file at the root of the archive, then build the archive. Lesson 5, "Testing Techniques" explains the reason for including the WSDL file.

1. Select **Project>Add to Project>File** from the menu.
2. In the Add to Project dialog, find the **src** directory under the project root, highlight **CalculatorImplWS.wsdl**, and click **Open**.

3. In the second Add to Project dialog, select **Add the file to the root of the archive** and click **OK**.



4. In Workbench, select **Project>Build and Archive** from the menu to create a deployable archive for your project.



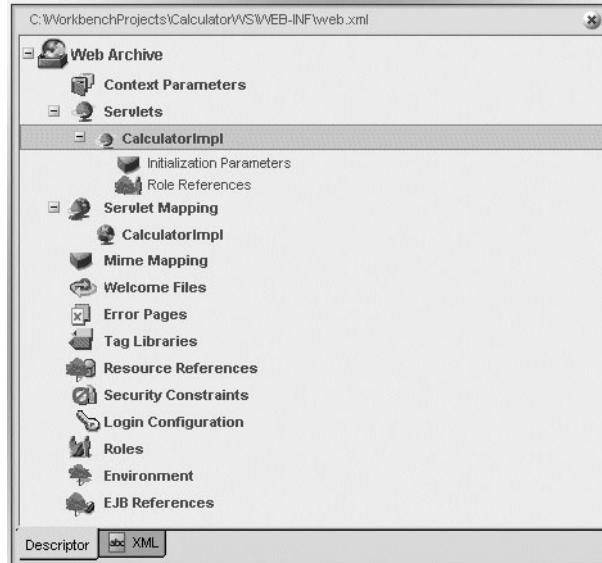
EXERCISE 2-8: Examine the deployment descriptor

The wizard inserts information about the main servlet for the Web Service into the deployment descriptor. In this exercise you'll take a look at that information so you'll know where to find and change it if you ever need to.

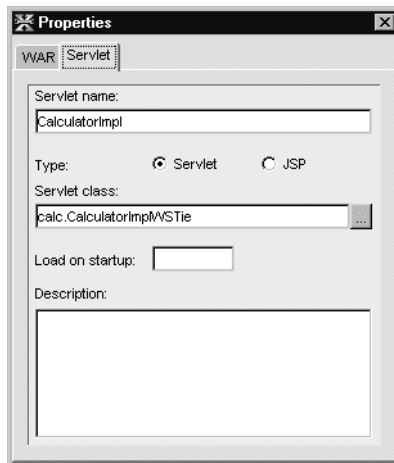
1. In the Navigation Pane, right-click the project file **Calculator.spf** and select **Open Deployment Descriptor** from the popup menu.

NOTE You can also find web.xml in any Source or Archive view and double-click it to open it.

If Workbench displays the **Select Build Option** dialog, accept the defaults and click **OK**. Workbench opens web.xml in the Edit Pane. The editor displays the Descriptor tab, showing the types of information the descriptor can include.

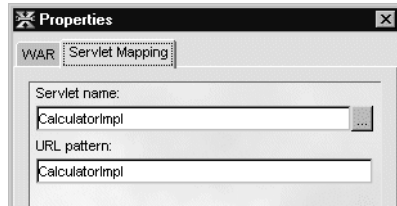


2. Notice the **CalculatorImpl** item under the **Servlets** heading. It was added by the wizard.
3. Right-click **CalculatorImpl** and select **Properties** from the popup menu. The property sheet displays the deployment properties for the servlet.



4. Notice that the value for **Servlet class** is **calc.CalculatorImplWSTie**. This is the class that will run when the Web Service is invoked.
5. Back in the Edit Pane, find and highlight the **CalculatorImpl** item in the **Servlet Mapping** section.

The property sheet now displays the mapping properties.




6. Notice that the value for **URL pattern** is CalculatorImpl, the same as the servlet's name. You will use this value in the URL that accesses the Web Service.
7. Close the deployment descriptor by clicking the button in the upper-right corner of the editor or selecting **File>Close** from the menu.


Deploying the project

If you've done another Workbench tutorial, most of your deployment setup has already been done. This exercise gives you the main steps and provides the project-specific information you'll need to deploy this project. For detailed deployment instructions for all the supported servers, see Workbench Deployment Instructions.



EXERCISE 2-9: Deploy the project

1. If you haven't created a profile for your server, select **Edit>Profiles** from the menu and create one now.
 -  For information, see the server profile procedure in the deployment instructions.
2. Use the following information to create the server-specific part of the deployment process. For most J2EE servers, the server-specific deployment information is in a separate file, usually in XML format. For some servers, you need to add it to your project so that it is built into the archive.

 For more information and exercises with detailed steps, select the section for your server in the deployment instructions.

Server	What to do	What to specify
SilverStream	Create a SilverStream deployment plan. In the Deployment Plan Editor, set values on the property sheet for the Web Archive item.	<p>Enabled — True</p> <p>Deployed object name — Calculator</p> <p>Server Profile — Select the profile you defined from the dropdown list box</p> <p>Session timeout — 5 minutes, the default</p> <p>URLs — Calculator, the default</p>
Sun Reference Implementation	Create a runtime deployment descriptor called sun-j2ee-ri.xml with the content at right. Put it in a directory called META-INF and add the file to the project.	<pre><?xml version="1.0" encoding="Cp1252"?> <j2ee-ri-specific-information> <server-name></server-name> <rolemapping /> <web> <display- name>Calculator</display-name> <context- root>Calculator</context-root> </web> </j2ee-ri-specific-information></pre>
Jakarta Tomcat	—	—
BEA WebLogic	Create a WebLogic descriptor called weblogic.xml with the content at right. Add it to the project in the WEB-INF directory.	<pre><!DOCTYPE weblogic-web-app PUBLIC "-//BEA Systems, Inc.//DTD Web Application 6.0//EN" "http://www.bea.com/servers/wls6 10/dtd/ weblogic-web-jar.dtd"> <weblogic-web-app> <description> Calculator Web Service </description> <weblogic-version> </weblogic-version> </weblogic-web-app></pre>

Server	What to do	What to specify
IBM WebSphere	—	—
Oracle9iAS	—	—

3. Specify deployment settings for your server by selecting **Project>Deployment Settings** from the menu.

On the **Server Profiles** tab, select the server profile you defined above. If you have a secure server, specify values for **User name** and **Password**.

On the **Deployment Info** tab, specify additional application-specific information, as follows.

NOTE For these tutorials, do not check **Enable Rapid Deployment**. For information on how to use rapid deployment with your server, see Archive Deployment in the *Tools Guide*.

Server	Option and Value
SilverStream	<p>SilverStream Deployment Plan — Select the plan you defined in Step 2</p> <p>Overwrite existing deployment — Selected</p> <p>Verbosity — 3</p> <p>Ignore compile errors — Not selected (if JSP pages don't compile successfully during deployment, don't deploy the archive)</p>
Sun Reference Implementation	—
Jakarta Tomcat	—
BEA WebLogic	WebLogic Application Name — Calculator; used in the URL for accessing the Web application

Server	Option and Value
IBM WebSphere	Node Name — Leave blank or specify a node you've set up on your server
Oracle9iAS	Deployment Name — Calculator; used in the URL for accessing the Web application Target Path — Leave blank or specify a path you've set up on your server Website Name — Accept the default value or specify a name you've set up on your server



For more details, select the section for your server in the deployment instructions.

4. Click **Deploy** in the Deployment Settings dialog.

OR

Click **OK** in Deployment Settings and select **Project>Deploy Archive** from the menu. Workbench displays progress messages, errors, and warnings on the Deploy tab of the Output Pane.

TIP For most server types, full deployment will fail if your server is not running. For some servers you need to restart after deployment. For details, see the section for your server in the deployment instructions.

Updating the J2EE server's classpath

Before running the Web Service, there is one more thing to do. You must make sure that the deployed WAR has runtime access to the following archives required by jBroker Web:

- **jbroker-web.jar**, which contains the jBroker Web API classes
- **jaxrpc-api.jar** and **saaj-api.jar**, which contain the Java API classes for XML-based RPC and SOAP processing
- **xerces.jar** or another XML parser

How you set up this access depends on the type of J2EE server you use:

If you deployed to one of the following servers, you must add the required JARs to the server's classpath. (Consult your server documentation to learn about adding to the classpath.)

- BEA WebLogic
- IBM WebSphere

- Jakarta Tomcat
- Oracle9i

If you deployed to the **SilverStream eXtend Application Server**, you don't need to add those JARs to the server's classpath (the fact that they are in the WEB-INF/lib directory in the WAR is sufficient for the SilverStream server).

Testing the Web Service

The Web Service Wizard generates a Java class for testing the Web Service. After you make a few modifications to the template code, you can run the program to see what happens.



EXERCISE 2-10: Edit the test client code

1. In the Navigation Pane, find **CalculatorImplWSSClient.java** and double-click it to open it in the editor. In Source Layout, it's in the src/calc directory; in Archive Layout, it's in WEB-INF/classes/calc.
2. In the **process()** method, replace the four commented **System.out.println()** statements with this code. **Do not** remove the call to **getRemote()**.

This new code gets arguments from the command line (or uses default values) and calls the CalculatorImpl methods.

```
double x, y;
if (args.length == 2)
{
    x = new Double(args[0]).doubleValue();
    y = new Double(args[1]).doubleValue();
}
else
{
    x = 4.0;
    y = 5.0;
}
System.out.println("Add      = " + remote.add(x, y));
System.out.println("Divide   = " + remote.divide(x, y));
System.out.println("Multiply = " + remote.multiply(x, y));
System.out.println("Subtract = " + remote.subtract(x, y));
```

3. Select **Project>Compile** from the menu to save and compile the file.
4. Close the file.



EXERCISE 2-11: Test the Web Service with the generated client

1. Select **Project>Run Web Service Client Class** from the menu.

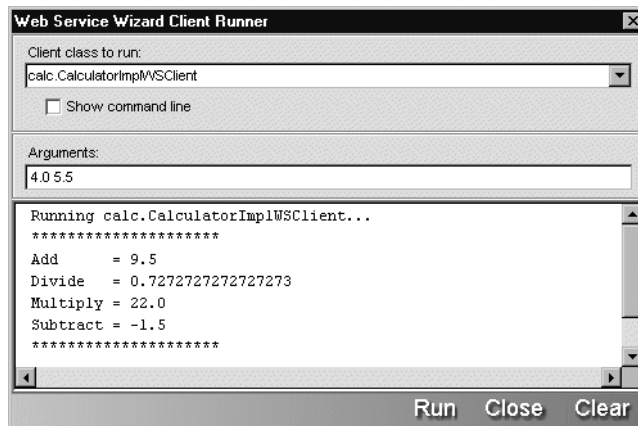
The selection list in the Web Service Wizard Client Runner window displays the test client class. If your project included other compiled classes with main() methods, they would be listed too.

2. In the **Arguments** text box, type two numbers, which are the input for the Calculator's arithmetic operations. For example, you might type:

4.0 5.5

3. Click **Run**.

The output from the System.out.println() methods displays in the output box.



4. Click **Close** when you are done.

Congratulations. You've successfully deployed and tested a Web Service.

Summary of what you've done

Developing the application In this lesson you built and deployed a WAR for a Web Service that provides several methods for basic arithmetic. You edited the code of the generated client program and ran the client to test the Web Service.

Using Workbench tools You used these tools in Workbench:

- New Project Wizard (File>New Project)

- Add to Project menu item (Project>Add to Project)
- Project Settings dialog (Project>Project Settings)
- Web Service Wizard (File>New, Web Services tab)
- Deployment tools (Open Deployment Descriptor on project popup menu, Edit>Profiles, Project>Deployment Settings, Project>Deploy Archive)
- Web Service Wizard Client Runner window (Project>Run Web Service Client Class)

Next lesson In the next lesson you will learn about generating client code from a WSDL file that describes a Web Service.

3

Creating a Client Application for a Web Service

What you will learn

When working with Web Services, there are two basic roles:

- The service **provider** who writes and deploys a service
- The service **consumer** who writes a client application that calls the methods offered by the service

In Lesson 2, “Creating a Web Service” you played the role of provider and deployed the Calculator Web Service. In this lesson you’ll be a service consumer and use the Web Service Wizard to generate code that calls the Calculator Web Service.

This lesson uses WSDL generated in Lesson 2, “Creating a Web Service” as its starting point. Although much of the code you need was already generated in that lesson, this lesson will proceed as if you had no source code for the Web Service, only a description file in Web Services Description Language (WSDL) format.

You will learn about:

- Getting information about a Web Service
- Setting up your project
- Generating client code from WSDL

What you will do

1. Set up a project directory and get the WSDL file
2. Create a new project
3. Set up a classpath for building the project
4. Generate client code from WSDL
5. Edit the test client code
6. Test the Web Service with the generated client

How long will it take? About 15 minutes

NOTE This lesson assumes you completed Lesson 2, “Creating a Web Service” and deployed the Calculator Web Service. When you test this project, the J2EE application server where the Calculator Web Service is deployed needs to be running.

Getting information about a Web Service

Web Services Description Language (WSDL) is a standard way to exchange information about a deployed Web Service. A WSDL file is an XML document that specifies the methods, data types, and URL of the Web Service. It allows the service to be described in an abstract, reusable way.

There are several scenarios for getting a WSDL file. You might:

- Get a WSDL file directly from a vendor who is deploying a service you want to use—for example, from a Web page or via e-mail
- Define a WSDL specification for a service jointly with business partners
- Use the Registry Manager to download a WSDL file from a public registry

Once you have a WSDL file for a Web Service, you can use Workbench’s Web Service Wizard to generate client code that invokes the service. The generated files include a remote interface, service classes, a stub, and a client program for testing.

Calculator Web Service In this lesson imagine that you got a WSDL file for the Calculator Web Service from another developer or business. In reality, you generated it in Lesson 2, “Creating a Web Service”.

Setting up your project

The client program you will build is a simple Java program, not a J2EE application stored in an archive. In Workbench you have to choose an archive type, so you’ll choose a JAR project. When you run the application, you can use either the command line or the Client Runner window. You won’t need to build and deploy an archive—just compile the files.



EXERCISE 3-1: Set up a project directory and get the WSDL file

1. Using your operating system tools, create a root directory for your project called **CalculatorClient**. You can put it at the root level of your disk drive or in a subdirectory of your choosing. The sample paths in this tutorial assume you created CalculatorClient in the WorkbenchProjects directory. On Windows, it would look like this:

```
c:\WorkbenchProjects\CalculatorClient
```
2. Copy the file **CalculatorImplWS.wsdl** to the CalculatorClient directory. You’ll find this file in the src directory of the project for Lesson 2, “Creating a Web Service”—for example, **c:\WorkbenchProjects\CalculatorWS\src**.

NOTE You can also get this file from *Workbench-install-directory\docs\tutorial\TutorialFiles\webservices*. If you do, **it is important** to open the file and edit the URL in the soap:address element at the end of the file to specify the URL where the Calculator Web Service is deployed.



EXERCISE 3-2: Create a new project

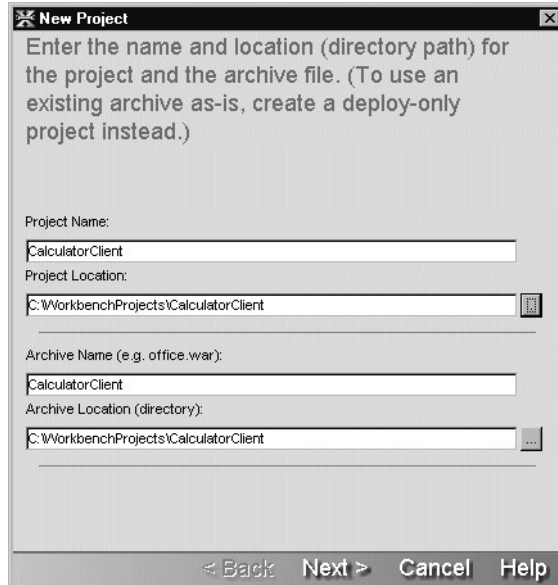
In this exercise you will start Workbench and use the New Project Wizard to create a project for a client application that uses the Calculator Web Service.

1. Start Workbench. You can use the SilverStream eXtend Workbench shortcut on the Windows Start menu.
OR
If Workbench is already running and a project is open, select **File>Close Project** from the menu. If prompted to close open files, click **Yes**.
2. Select **File>New Project** from the menu.
3. In the New Project Wizard, select **JAR** and then click **OK**.



4. On the next panel, in the Project Name text box type **CalculatorClient**.

5. Click the ellipses beside the Project Location text box and select the **CalculatorClient** directory you created in EXERCISE 3-1: “Set up a project directory and get the WSDL file”. When you click **OK**, the rest of the panel is filled in automatically.



6. Click **Next**, check the project specifications on the final panel, then click **Finish**. In the Navigation Pane, the Project tab displays the new project.



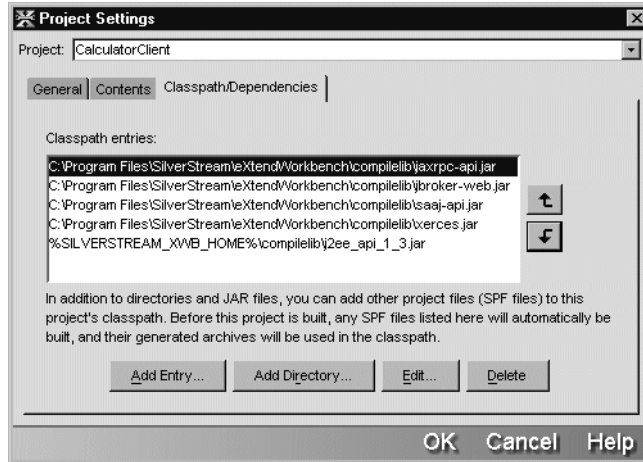
EXERCISE 3-3: Set up a classpath for building the project

In this exercise you will use the Project Settings dialog to specify a compile-time classpath. For Web Services and Web Service clients, the classpath needs to include `jbroke-er-web.jar` and some supporting JARs.

1. With your project open, choose **Project>Project Settings** from the menu.
2. Select the **Classpath/Dependencies** tab.
3. Click the **Add Entry** button.
4. In the Add to Classpath dialog, navigate to the **Workbench-install-directory/compilelib** directory. Highlight the following files and click **Open** then **OK**.
 - `jaxrpc-api.jar`
 - `jbroke-er-web.jar`

- saaj-api.jar
- xerces.jar

Now the Classpath/Dependencies tab should look something like this:



5. Click **OK** to close the Project Settings dialog.

Generating client code from WSDL

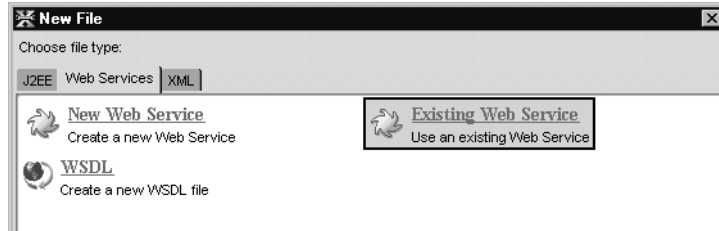
In Lesson 2, “Creating a Web Service”, you created a new Web Service starting with a Java class. Here you’ll start with a WSDL file that represents an existing Web Service.



EXERCISE 3-4: Generate client code from WSDL

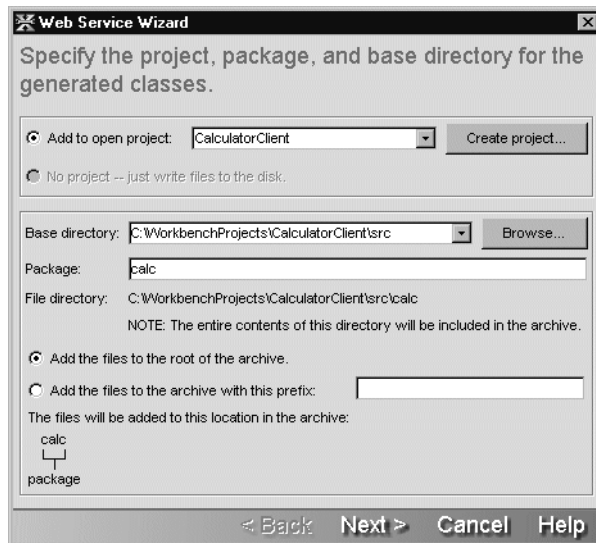
1. With your project open in Workbench, select **File>New** from the menu.
2. In the New File dialog, click the **Web Services** tab, select **Existing Web Service**, and click **OK**.

3 Creating a Client Application for a Web Service



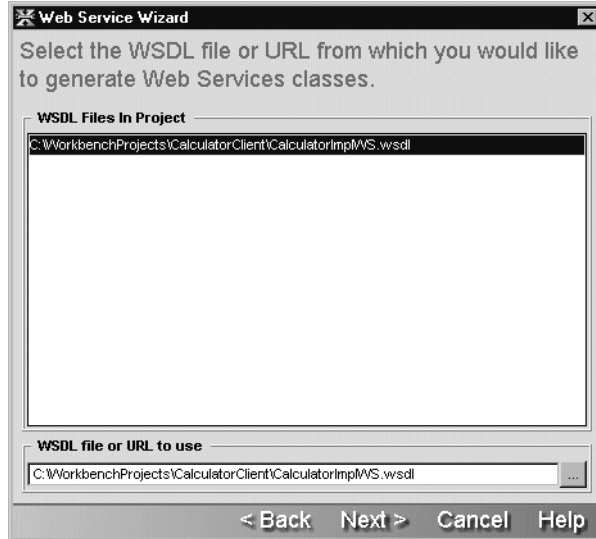
Workbench displays the project location panel of the Web Service Wizard.

3. Fill in the panel as shown below. The only value you should have to specify is the **calc** package.



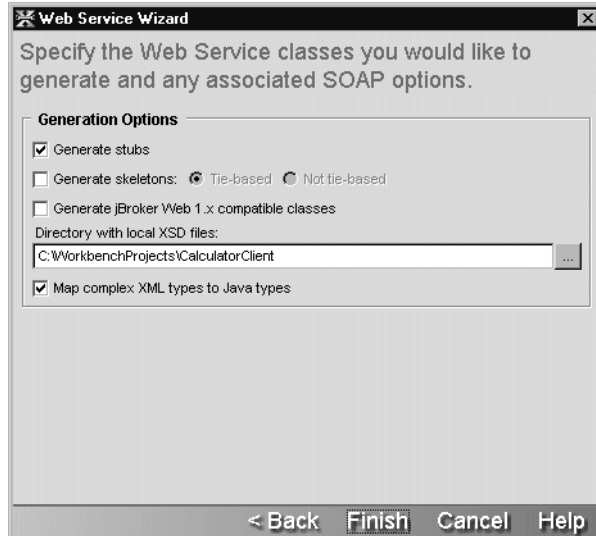
Once you click **Next**, Workbench displays the WSDL file selection panel of the Web Service Wizard. The file **CalculatorImplWS.wsdl** that you saved in the project root directory appears in the **WSDL Files in Project** list box.

4. Highlight **CalculatorImplWS.wsdl** so that it appears in the **WSDL file or URL to use** box. Then click **Next**.



The class generation and SOAP options panel displays.

5. Examine the settings on this panel (you don't need to change any of them).



6. Click **Finish**.

Wizard results

After you run the wizard, a new directory **src** is added to your project. The **calc** package directory under it contains several new files. The **Generate stubs** option produces these files for use in a client application:

File	Description
CalculatorImplWS.java	A remote interface that has declarations for the methods specified in the WSDL file. It extends <code>java.rmi.Remote</code> . Each of the methods throws <code>RemoteException</code> .
CalculatorImplWSService.java	A Service interface used by JAX-RPC clients to obtain the stub for the Web Service. You should never need to modify this class.
CalculatorImplWSServiceImpl.java	A Service implementation class that handles instantiation of the stub (<code>CalculatorImplWSBinding_Stub</code>). You should never need to modify this class.
CalculatorImplWSBinding_Stub.java	A <code>JBroker Web</code> class that processes SOAP messages on the client. You should never need to modify this class.
CalculatorImplWSClient.java	A standalone Java program for accessing the Web Service. After you edit the code, use it to call methods of the Web Service.
DivideFault.java	An exception class thrown by the <code>divide()</code> method in the remote interface for this project. This file is specific to this project.
DivideFaultMarshaler.java	A marshaler that serializes and deserializes the <code>DivideFault</code> data type when it needs to be sent in a SOAP message. This file is specific to this project.

Editing and testing the client application

Code to instantiate the stub The generated client code obtains the stub by calling a method of the Service object (which is obtained via JNDI). The code looks like this:

```
public CalculatorImplWS getRemote(String[] args) throws Exception
{
    InitialContext ctx = new InitialContext();
```



```
String lookup = "xmlrpc:soap:calc.CalculatorImplWSService";
CalculatorImplWSService service = (CalculatorImplWSService)ctx.lookup(lookup);
CalculatorImplWS remote = (CalculatorImplWS)service.getCalculatorImplWSPort();

return remote;
}
```

About the binding When you created the Calculator Web Service in Lesson 2, “Creating a Web Service”, you specified the binding—the URL for the Web Service—according to where you were going to deploy the Web Service. For the Calculator client, the wizard gets that binding from the WSDL and includes it in the generated stub.

If the URL changes, you can override the binding in the stub like this:

```
public CalculatorImplWS getRemote(String[] args) throws Exception
{
    InitialContext ctx = new InitialContext();

    String lookup = "xmlrpc:soap:calc.CalculatorImplWSService";
    CalculatorImplWSService service = (CalculatorImplWSService)ctx.lookup(lookup);
    CalculatorImplWS remote = (CalculatorImplWS)service.getCalculatorImplWSPort();

    ((javax.xml.rpc.Stub)remote)._setProperty("javax.xml.rpc.service.endpoint.address",
        "http://www.myserver.com:80/Calculator/CalculatorImpl");

    return remote;
}
```

For now, the original binding is what you want. Before you run the test client, all you need to do is edit the calls to the Web Service. You did these same steps when you tested the Web Service in Lesson 2, “Creating a Web Service”.



EXERCISE 3-5: Edit the test client code

In this exercise you’ll use the same client test code as you used for testing the Web Service in Lesson 2, “Creating a Web Service”.

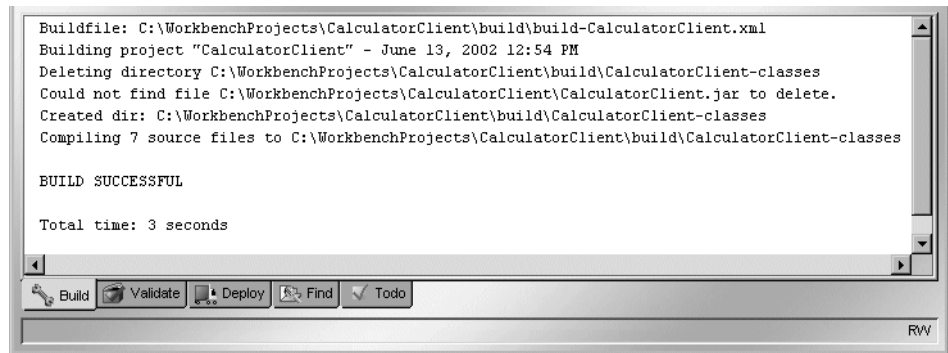
1. In the Navigation Pane, find **CalculatorImplWSClient.java** and double-click it to open it in the Edit Pane. In Source Layout, it’s in the `src/calc` directory.
2. In the **process()** method, replace the four commented **System.out.println()** statements with the following code. **Do not** remove the call to `getRemote()`.

This new code gets arguments from the command line (or uses default values) and calls the four Calculator methods.

```
double x, y;
if (args.length == 2)
{
    x = new Double(args[0]).doubleValue();
    y = new Double(args[1]).doubleValue();
}
else
{
    x = 4.0;
    y = 5.0;
}
System.out.println("Add      = " + remote.add(x, y));
System.out.println("Divide   = " + remote.divide(x, y));
System.out.println("Multiply = " + remote.multiply(x, y));
System.out.println("Subtract = " + remote.subtract(x, y));
```

3. Save the file.
4. Select **Project>Build** from the menu.

The Build tab of the Output Pane should report a successful build.



EXERCISE 3-6: Test the Web Service with the generated client

1. Select **Project>Run Web Service Client Class** from the menu.

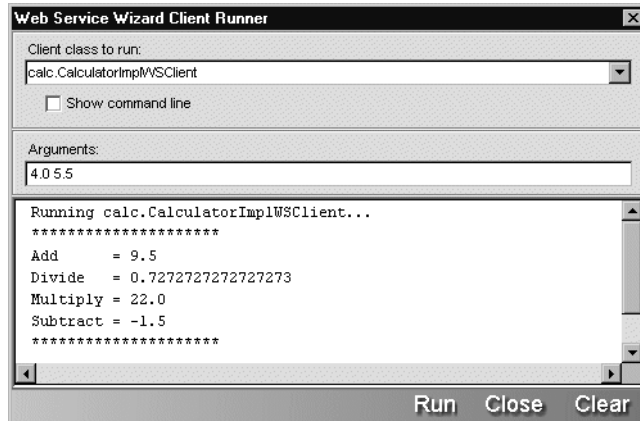
The selection list in the Web Service Wizard Client Runner window displays the test client class. If your project included other compiled classes with main() methods, they would be listed too.

2. In the **Arguments** text box, type two numbers—for example:

```
4.0 5.5
```

3. Click **Run**.

The output from the `System.out.println()` methods displays in the output box.



4. Click **Close** when you are done.

Congratulations. You've successfully invoked the publicly available Calculator Web Service.

Summary of what you've done

Developing the application In this lesson you used a Web Service description (WSDL) file to generate code that accesses a Web Service.

Using Workbench tools You used these tools in Workbench:

- New Project Wizard (File>New Project)
- Project Settings dialog (Project>Project Settings)
- Web Service Wizard (File>New, Web Services tab)
- Web Service Wizard Client Runner window (Project>Run Web Service Client Class)

Next lesson In the next lesson you will learn about building a Web application as a client for the Calculator Web Service.

4

Using Web Services in a J2EE Web Application

What you will learn

This lesson teaches you how to create a J2EE Web application that is a client of a Web Service. The Web application is a single JSP page. The JavaBean for the page has methods that instantiate a remote object and call the Calculator Web Service from Lesson 2, “Creating a Web Service”.

You will learn about:

- Defining a WAR project for the Web Service client application
- Adding Web Service client code to the project
- Creating a form that calls the Calculator Web Service
- Deploying and testing the WAR

What you will do

1. Create a new project
2. Add the jBroker Web libraries to the project
3. Generate the client code for the Calculator Web Service
4. Create a new JSP page
5. Edit the JSP page
6. Create a second JSP page to include in `magicnumber.jsp`
7. Write a JavaBean to process the form
8. Build the archive
9. Edit the deployment descriptor
10. Deploy the project
11. Test the Calculator Client application

How long will it take? About 20 minutes

NOTE You need to run your J2EE application server to deploy the WAR you create in this lesson. The Web Service you deployed in Lesson 2, “Creating a Web Service” must also be running.

Defining a WAR project for the Web Service client application

In this section you'll create a WAR project for a Web application whose pages call the Calculator Web Service.

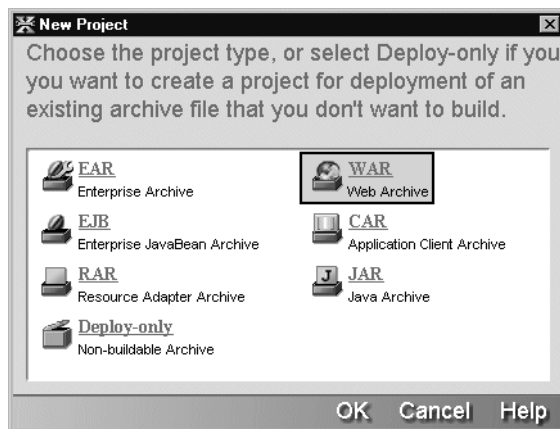
In the previous lesson you created the project directory, then defined the project in Workbench. This time you'll let Workbench create the project root directory.



EXERCISE 4-1: Create a new project

In this exercise you will start Workbench and use the New Project Wizard to create a project for a Web application.

1. Start Workbench. You can use the SilverStream eXtend Workbench shortcut on the Windows Start menu.
OR
If Workbench is already running and a project is open, select **File>Close Project** from the menu. If prompted to close open files, click **Yes**.
2. Select **File>New Project** from the menu.
3. In the New Project Wizard, select **WAR** and then click **OK**.



4. In the Project Name field, type **CalcWARClient**.

5. Specify the full path for a project root directory called **CalcWARClient** in the **Project Location** text box.

You can type something like **c:\WorkbenchProjects\CalcWARClient**, or you can click the ellipses to select a parent directory in the Choose Directory dialog. Then you can type the new directory name in the Project Location text box after the selected directory.

As you type, you see the rest of the dialog filled in automatically.

6. In the Project J2EE Version field, specify **J2EE 1.2 (WAR 2.2)** so your application will run on any server that supports J2EE 1.2 or 1.3.

New Project

Enter the name and location (directory path) for the project, the archive file, and the deployment descriptor and select the desired J2EE version. (To use an existing archive as-is, create a deploy-only project instead.)

Project Name:
CalcWARClient

Project Location:
C:\WorkbenchProjects\CalcWARClient

Archive Name (e.g. office.war):
CalcWARClient

Archive Location (directory):
C:\WorkbenchProjects\CalcWARClient

Deployment Descriptor Name:
web.xml

Deployment Descriptor Location:
C:\WorkbenchProjects\CalcWARClient\WEB-INF

Project J2EE Version:
J2EE 1.2 (WAR 2.2)

< Back Next > Cancel Help

7. Click **Next**.
8. When the wizard asks if it should create the project root and WEB-INF directories, click **Yes**.

The wizard summarizes the project information.

9. Click **Finish**.

In the Navigation Pane, the Project tab displays the new project.



EXERCISE 4-2: Add the jBroker Web libraries to the project

The Web Service client uses classes in `jbroker-web.jar` and supporting JARs for SOAP message processing. In this exercise you will add these JARs to the archive for runtime access and to the project classpath for compile-time access.

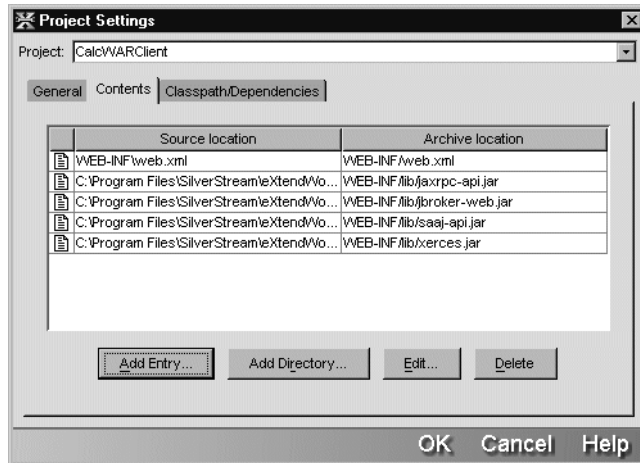
1. With your project open, select **Project>Project Settings** from the menu.
2. Select the **Contents** tab and click the **Add Entry** button.
3. In the Select Contents dialog, navigate to the directory *Workbench-install-directory/compilelib*, then highlight the following files and click **Open**:
 - `jaxrpc-api.jar`
 - `jbroker-web.jar`
 - `saaj-api.jar`
 - `xerces.jar`

The Add to Project dialog will prompt you for information about each file, one at a time.

4. When you're prompted about `jaxrpc-api.jar`, select **Add the file to the archive at this location**. In the text box, type **WEB-INF/lib/jaxrpc-api.jar**. Then click **OK**.
5. When you're prompted about `jbroker-web.jar`, select **Add the file to the archive at this location**. In the text box, type **WEB-INF/lib/jbroker-web.jar**. Then click **OK**.
6. When you're prompted about `saaj-api.jar`, select **Add the file to the archive at this location**. In the text box, type **WEB-INF/lib/saaj-api.jar**. Then click **OK**.

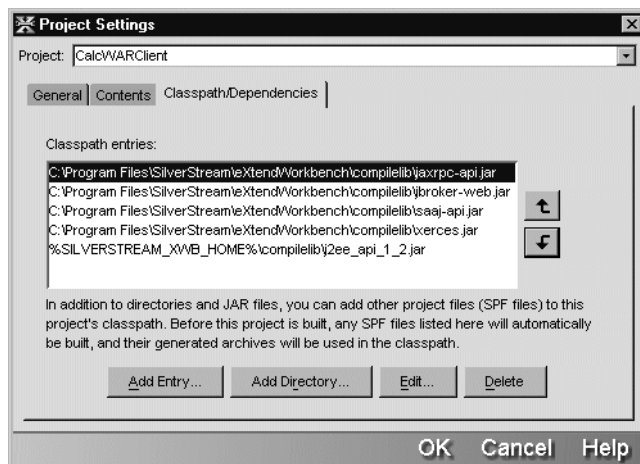
7. When you're prompted about `xerces.jar`, select **Add the file to the archive at this location**. In the text box, type **WEB-INF/lib/xerces.jar**. Then click **OK**.

The WEB-INF/lib directory of the archive will now include these JARs.



8. Select the **Classpath/Dependencies** tab and click the **Add Entry** button.
9. In the Add to Classpath dialog, find the directory *Workbench-install-directory/compilelib* again, then highlight the following files and click **Open** then **OK**.
- `jaxrpc-api.jar`
 - `jbroker-web.jar`
 - `saaj-api.jar`
 - `xerces.jar`

The Classpath/Dependencies tab should look something like this:



10. Click **OK** to close the Project Settings dialog.

Adding Web Service client code to the project

There are several classes that the Calculator WAR Client needs for accessing the Calculator Web Service:

- CalculatorImplWS.java
- CalculatorImplWSService.java
- CalculatorImplWSServiceImpl.java
- CalculatorImplWSBinding_Stub.java
- DivideFault.java
- DivideFaultMarshaler.java

If you did Lesson 2, “Creating a Web Service” or Lesson 3, “Creating a Client Application for a Web Service”, the client files already exist in a calc package in those projects. Those lessons also explain what each file does. Although you could copy the files to this project or add them from their current location, instead you’ll use the WSDL file from Lesson 2, “Creating a Web Service” to generate them again—it’s quick and easy to do.



EXERCISE 4-3: Generate the client code for the Calculator Web Service

In this exercise you'll generate the client code from the WSDL file for the Web Service. This exercise is a synopsis of the same steps you did in Lesson 3, "Creating a Client Application for a Web Service". For pictures and information about the results of the Web Service Wizard, see that lesson.

1. Using your system tools, copy the file **CalculatorImplWS.wsdl** to the **CalcWARClient** directory. You'll find this file in the **src** directory of the project for Lesson 2, "Creating a Web Service"—for example, **c:\WorkbenchProjects\CalculatorWS\src**.

TIP If you didn't do Lesson 2, "Creating a Web Service" and will use someone else's deployed Calculator Web Service, you can get the file from the **Workbench-install-directory\docs\tutorial\TutorialFiles\webservices** directory. If you use this file, **it is important** to open it and edit the URL in the **soap:address** element at the end of the file to specify the URL where the Calculator Web Service is deployed.

2. In Workbench, select **File>New** from the menu.
3. In the New File dialog, click the **Web Services** tab, select **Existing Web Service**, and click **OK**.

Workbench displays the project location panel of the Web Service Wizard.

4. Specify the package **calc** and click **Next**.
5. When the WSDL file selection panel displays, highlight **CalculatorImplWS.wsdl** and click **Next**.

The class generation and SOAP options panel displays.

6. Examine the settings on this panel (you don't need to change any of them), then click **Finish**.

Your project should now contain the client code for calling the Calculator Web Service.

Creating a form that calls the Calculator Web Service

A JSP page with a form uses a companion JavaBean to manage the data in the form fields. Properties of the JavaBean store the entered values and make them available to methods in the bean for further processing.

The simple application in this lesson uses that approach. When the user submits the form with data, the associated JavaBean stores the submitted values. When the JSP page is redisplayed, it tests whether data was submitted. If so, it calls a method of the JavaBean that invokes the Calculator Web Service. When the application successfully calculates the “magic number”, a second JSP fragment is included in the original page to display the result.

With the exercises in this section, you’ll create these files:

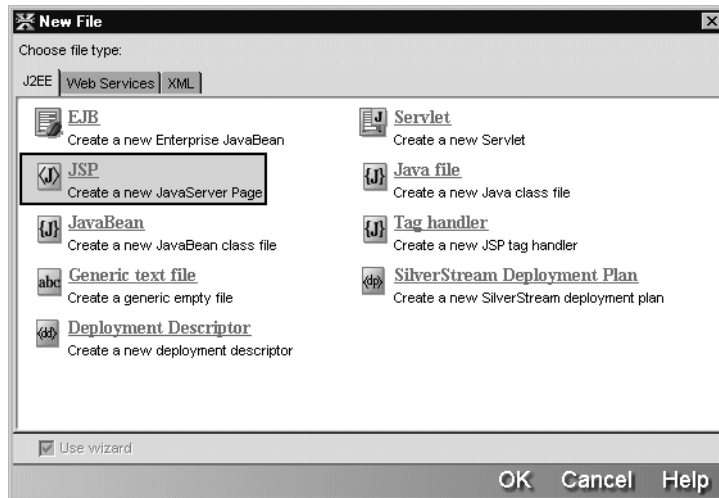
File	Description
magicnumber.jsp	The main page of the application with an input form
MagicNumberBean.java	JavaBean that handles the data from magicnumber.jsp
calcnnumber.jsp	JSP fragment that displays the calculated result via an include directive in magicnumber.jsp



EXERCISE 4-4: Create a new JSP page

In this exercise you will use the JSP Wizard to create a new page.

1. In Workbench, select **File>New** from the menu.



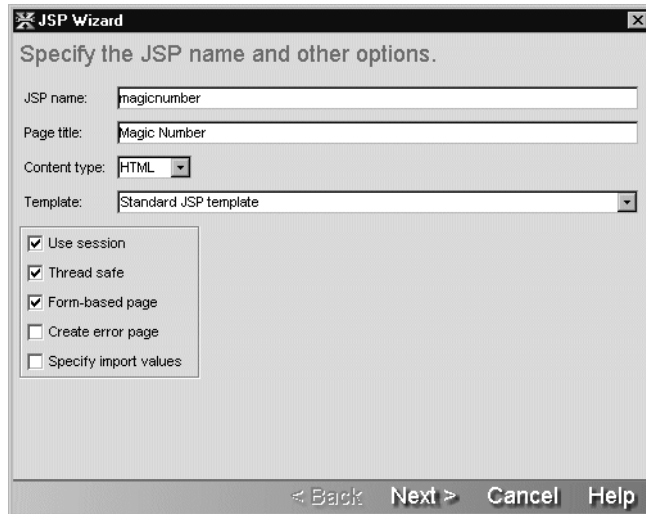
2. In the New File dialog, select **JSP** and click **OK**.

Workbench displays the JSP Wizard.

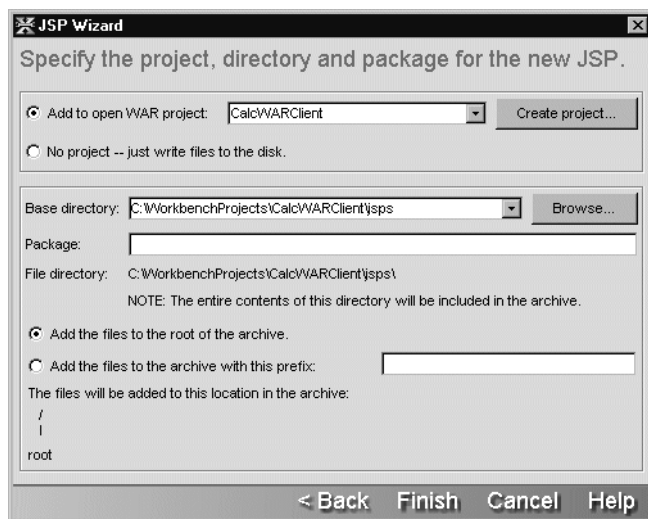
3. Fill out the first panel of the wizard with this information:

Option	Value
JSP name	magicnumber (don't specify the jsp extension)
Page title	Magic Number
Content type	HTML (the default)
Template	Standard JSP template (the default)
Other options	Use session, Thread safe, Form-based page

Now the first panel looks like this:



4. Click **Next**.
5. On the second panel, leave **Add to open WAR project** selected.
6. Specify where to put the file in the project and the archive:
- For **Base directory**, specify the full path for a new **jsps** directory—for example, `c:\WorkbenchProjects\CalcWARClient\jsps`. You can select another project directory path from the dropdown list and edit it.
 - For **Package**, leave it blank. In this project the JSP pages are at the root of the archive.
 - Leave **Add the files to the root of the archive** selected.



7. Click **Finish**.
8. When the JSP Wizard dialog reports that it is done creating the JSP page, click **OK**.
The new file is open in the Edit Pane. In the Navigation Pane you can see `magicnumber.jsp` in the `jsps` directory of the Source layout and at the archive root of the Archive Layout.



EXERCISE 4-5: Edit the JSP page

In this exercise you'll write the HTML and JSP code for a form that provides data for a calculation.

NOTE You can copy the code for this exercise from the file `CutAndPasteCode.txt` in the `Workbench-install-directory/docs/tutorial/TutorialFiles/webservices` directory.
OR

If you don't want to do these editing steps, you can use the correctly edited file `magicnumber-sample.jsp` in the same directory. Use your operating system tools to copy it to your project's `jsps` directory and rename it `magicnumber.jsp`, replacing the file you just created with the JSP Wizard.

1. With `magicnumber.jsp` open in the Edit Pane, add these lines after `</head>`:

```
<jsp:useBean id="magicnumber" class="com.client.MagicNumberBean"/>
<jsp:setProperty name="magicnumber" property="*/>
```

2. Add these lines between `<body>` and `</body>`, replacing the existing text:

```

<h1>Your Magic Number</h1>

<p>Your magic number changes every day.</p>

<form method="post">
  <table>
    <tr>
      <td>Your age</td>
      <td>
        <input type="text" name="age" value="<%= magicnumber.getAge() %>"
      >
    </td>
    </tr>
    <tr>
      <td>Day of month you were born</td>
      <td>
        <input type="text" name="birthday" value="<%=
magicnumber.getBirthday() %>" >
      </td>
    </tr>
    <tr>
      <td>Hour you went to bed last night</td><td>
        <input type="text" name="bedtime" value="<%=
magicnumber.getBedtime() %>" >
      </td>
    </tr>
    <tr>
      <td colspan="2" span="2">
        <input type="submit" name="Submit" value="Submit">
      </td>
    </tr>
  </table>
</form>

<%
  if (request.getParameter("age") != null )
  {
    magicnumber.calcNumber();
  }
  <%@ include file="calcnumber.jsp" %>
  <%
  }
  %>
  %>

```

3. Save and close the file.



EXERCISE 4-6: Create a second JSP page to include in `magicnumber.jsp`

In this exercise you'll create a JSP fragment that is included in `magicnumber.jsp` when there is a calculated result to display.

NOTE You can copy the code for this exercise from the file **CutAndPasteCode.txt** in the **Workbench-install-directory/docs/tutorial/TutorialFiles/webservices** directory.
OR

If you don't want to do these editing steps, you can use the correctly edited file **calcnnumber-sample.jsp** in the same directory. Use your operating system tools to copy it to the project's `jsps` directory and rename it **calcnnumber.jsp**.

1. In Workbench, select **File>New** from the menu.
2. In the New File dialog, select **JSP** and click **OK**.
Workbench displays the JSP Wizard.
3. For JSP name, specify **calcnnumber**. The rest of the values don't matter since you'll be replacing all the generated code.
4. Click **Next**.
5. On the second panel, leave **Add to open WAR project** selected.
6. Specify where to put the file in the project and the archive:
 - For **Base directory**, specify the full path for the `jsps` directory—for example, `c:\WorkbenchProjects\CalcWARClient\jsps`. You can use the dropdown list box or the Browse button to select it.
 - For **Package**, leave it blank. In this project the JSP pages are at the root of the archive.
 - Leave **Add the files to the root of the archive** selected.
7. Click **Finish**.
8. When the JSP Wizard dialog reports that it is done creating the JSP page, click **OK**.
The file is open in the Edit Pane. In the Navigation Pane you can see that `calcnnumber.jsp` has been added to the `jsps` directory in the Source layout and the archive root in the Archive layout.
9. Edit the file, replacing all the contents with this code:

```
<h2>Drumroll...</h2>

<table>
<tr>
<td>Your number is:</td>
<td><%= magicnumber.getMagicNumber() %></td>
</tr>
```



```
<tr>
  <td span="2">Did you expect a winning lottery number?</td>
</tr>
</table>
```

As you can see, the code is not a complete HTML page. It will be included in the other JSP page.

10. Save and close the file.



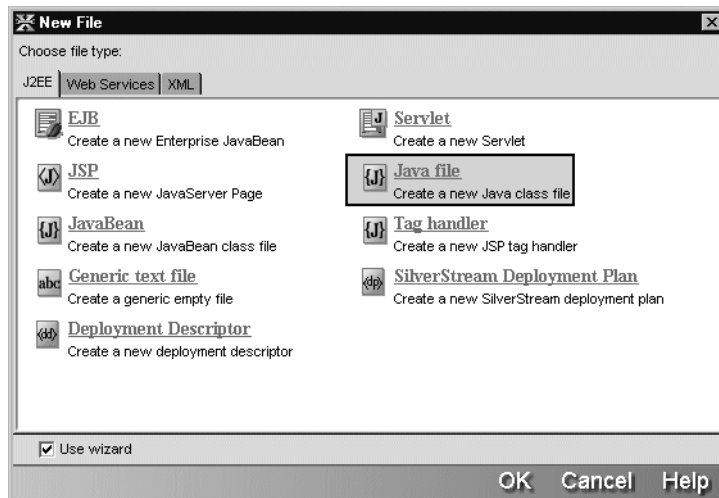
EXERCISE 4-7: Write a JavaBean to process the form

In this exercise you'll create a new Java source file by using the Java Class Wizard and then copy in the code for the JavaBean. (An alternative would be to use the JavaBean Wizard provided by Workbench. It is most useful when you're creating your own JavaBeans from scratch.)

NOTE You can copy the code for this exercise from the file **CutAndPasteCode.txt** in the **Workbench-install-directory/docs/tutorial/TutorialFiles/webservices** directory.
OR

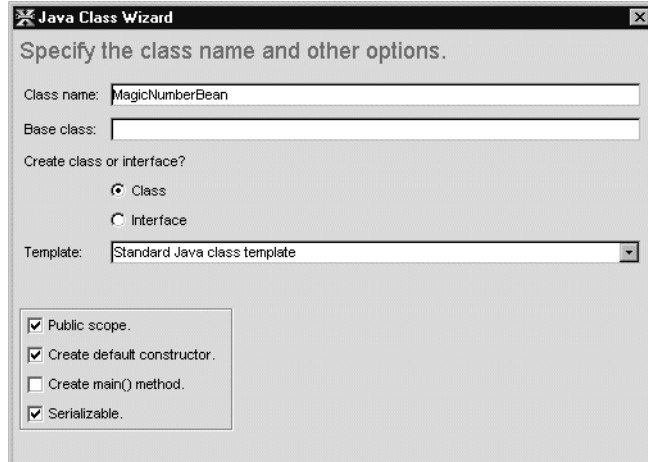
If you don't want to do these editing steps, you can use the correctly edited file **MagicNumberBean-sample.java** in the same directory. Use your operating system tools to create a directory called **com\client** under the **src** directory of your project, copy the file there, and rename it **MagicNumberBean.java**.

1. In Workbench, select **File>New** from the menu.

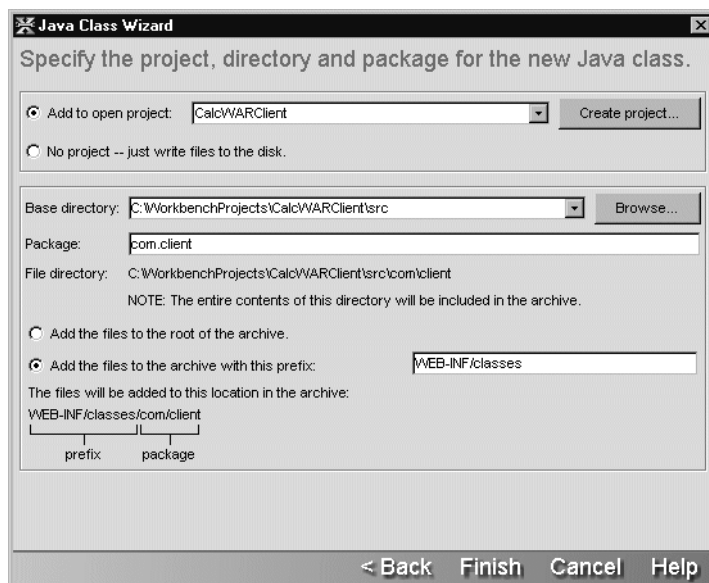


2. In the New File dialog, select **Java file** and click **OK**.
3. In the Java Class Wizard, specify these values:

Option	Value
Class name	MagicNumberBean (don't specify the java extension)
Base class	Leave blank
Create class or interface	Class (the default)
Template	Standard Java class template (the default)
Other options	Public scope, Create default constructor, Serializable



4. Click **Next**.
5. When the wizard prompts for interfaces to add, click **Next** to skip to the next panel.
6. When the wizard prompts for additional imports, click **Next** to skip to the next panel.
7. On the next panel, leave **Add to open project** selected.
8. Specify where to put the file in the project and the archive:
 - For **Base directory**, specify the full path for the **src** directory—for example, `c:\WorkbenchProjects\CalcWARClient\src`. You will find this path on the dropdown list.
 - For **Package**, specify `com.client`.
 - Select **Add the files to the archive with this prefix** and specify `WEB-INF/classes` as the prefix.



9. Click **Finish**.

10. When the Java Class Wizard dialog reports that it is done creating the new Java file, click **OK**.

The file `MagicNumberBean.java` is open in the Edit Pane.

11. In the Edit Pane, add these import statements after the package statement:

```
import javax.naming.InitialContext;
import calc.*;
```

12. Replace the constructor, which looks like this:

```
public MagicNumberBean()
{
    /** @todo: implement this constructor */
}
```

with these property variables, constructor, and getter and setter methods. The properties with their getter and setter methods correspond to fields in the form in `magicnumber.jsp`.

```
private int age=0;
private int birthday=0;
private int bedtime=0;
private double magicNumber=0.0;

public MagicNumberBean() { }
public int getAge() { return this.age; }
public void setAge(int age) { this.age=age; }
```

```

public int getBirthday() { return this.birthday; }
public void setBirthday(int day) { this.birthday=day; }

public int getBedtime() { return this.bedtime; }
public void setBedtime(int bedtime) { this.bedtime=bedtime; }

public double getMagicNumber() { return this.magicNumber; }
public void setMagicNumber(double num) { this.magicNumber=num; }

```

13. Before the final closing } for the class, add the calcNumber() and getCalculatorRemote() methods, which have the code for calling the Web Service.

```

public void calcNumber()
{
    double result=0;
    try
    {
        CalculatorImplWS remote = getCalculatorRemote();

        result = remote.add(age, birthday);
        if (result != 0)
        {
            result = remote.multiply(result, bedtime);
        }
        setMagicNumber(result);
    }
    catch (Exception _e)
    {
        System.out.println("*** Error calculating number ***");
        _e.printStackTrace();
    }
}

private CalculatorImplWS getCalculatorRemote() throws Exception
{
    InitialContext ctx = new InitialContext();

    String lookup = "xmlrpc:soap:calc.CalculatorImplWSService";
    CalculatorImplWSService service =
    (CalculatorImplWSService) ctx.lookup(lookup);
    CalculatorImplWS remote =
    (CalculatorImplWS) service.getCalculatorImplWSPort();

    return remote;
}

```

14. Save and close the file.

Deploying and testing the WAR

To deploy the application, you need to specify:

- Information in the deployment descriptor about the starting servlet or JSP page
- Information your server needs in the format it expects

You will do this next.

About the deployment descriptor

When you created the project, Workbench created an XML descriptor file appropriate to the type of archive you selected. For a WAR, the file is called **web.xml**.

When you open web.xml for editing, the Deployment Descriptor Editor shows the XML elements in an expandable tree structure. You can also look at the raw XML. The editor uses the project's compiled code to determine what to show, which is why you build the archive first. If it isn't already built, Workbench offers to build it for you.



EXERCISE 4-8: Build the archive

- In Workbench, select **Project>Rebuild All and Archive** from the menu to create a deployable archive for your project.



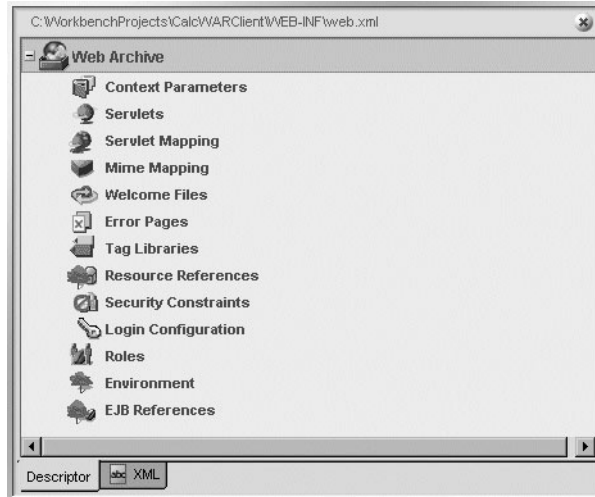
EXERCISE 4-9: Edit the deployment descriptor

In this exercise you'll identify the JSP page that is the entry point for the application.

1. In the Navigation Pane, right-click the project file **CalcWARClient.spf** and select **Open Deployment Descriptor** from the popup menu.

NOTE If Workbench displays the **Select Build Option** dialog, select **No, don't build now** and click **OK**. You can set options that cause this dialog to always or never display.

Workbench opens web.xml in the Edit Pane. The editor displays the Descriptor tab, showing the types of information the descriptor can include.



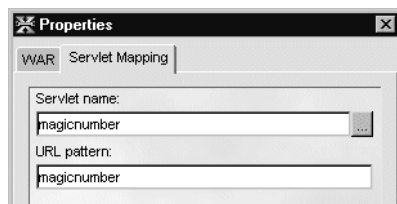
2. Right-click **Web Archive** and select **Properties** from the popup menu.
3. On the property sheet, specify **CalcWARClient** for Display Name.
4. In the Edit Pane, right-click **Servlets** and select **Add** from the popup menu.
5. On the property sheet, specify these values:

Option	Value
Servlet name	magicnumber
Type	JSP
JSP file	magicnumber.jsp



6. In the Edit Pane, right-click **Servlet Mapping** and select **Add** from the popup menu.
7. On the property sheet, specify these values:

Option	Value
Servlet name	magicnumber
URL pattern	magicnumber



8. In the Edit Pane, right-click **Welcome Files** and select **Add** from the popup menu.
9. On the property sheet, specify these values:

Option	Value
Welcome File	magicnumber.jsp



10. Save and close the deployment descriptor.

Deploying the project

If you've done the previous lessons, most of your deployment setup has already been done. This exercise gives you the main steps and provides the project-specific information you'll need to deploy this project. For detailed deployment instructions for all the supported servers, see Workbench Deployment Instructions.



EXERCISE 4-10: Deploy the project

- If you haven't created a profile for your server, select **Edit>Profiles** and create one now.
 For information, see the server profile procedure in the deployment instructions.
- Use the following information to create the server-specific part of the deployment process.
 For more information and exercises with detailed steps, select the section for your server in the deployment instructions.

Server	What to do	What to specify
SilverStream	Create a SilverStream deployment plan and set values on the property sheet for the Web Archive item.	<p>Enabled — True</p> <p>Deployed object name — CalcWARClient</p> <p>Server Profile — Select the profile you defined from the dropdown list box</p> <p>Session timeout — 5 minutes, the default</p> <p>URLs — CalcWARClient, the default</p> <p>Excluded JSPs — calcnumber.jsp</p>

Server	What to do	What to specify
Sun Reference Implementation	Create a runtime deployment descriptor called sun-j2ee-ri.xml with the content at right. Put it in a directory called META-INF and add the file to the project.	<pre><?xml version="1.0" encoding="Cp1252"?> <j2ee-ri-specific-information> <server-name></server-name> <rolemapping /> <web> <display- name>CalcWARClient</display- name> <context- root>CalcWARClient</context- root> </web> </j2ee-ri-specific-information></pre>
Jakarta Tomcat	—	—
BEA WebLogic	Create a WebLogic descriptor called weblogic.xml with the content at right. Add it to the project in the WEB-INF directory.	<pre><!DOCTYPE weblogic-web-app PUBLIC "-//BEA Systems, Inc.//DTD Web Application 6.0//EN" "http://www.bea.com/servers/wls6 10/dtd/ weblogic-web-jar.dtd"> <weblogic-web-app> <description> Calculator Client </description> <weblogic-version> </weblogic-version> </weblogic-web-app></pre>
IBM WebSphere	—	—
Oracle9iAS	—	—

- Specify deployment settings for your server by selecting **Project>Deployment Settings** from the menu.

On the **Server Profiles** tab, select the server profile you defined above. If you have a secure server, specify values for **User name** and **Password**.

On the **Deployment Info** tab, specify additional application-specific information, as follows.

NOTE For these tutorials, do not check **Enable Rapid Deployment**. For information on how to use rapid deployment with your server, see Archive Deployment in the *Tools Guide*.

Server	Option and Value
SilverStream	<p>SilverStream Deployment Plan — Select the plan you defined in Step 2</p> <p>Overwrite existing deployment — Selected</p> <p>Verbosity — 3</p> <p>Ignore compile errors — Not selected (if JSP pages don't compile successfully during deployment, don't deploy the archive)</p>
Sun Reference Implementation	—
Jakarta Tomcat	—
BEA WebLogic	WebLogic Application Name — CalculatorWARClient; used in the URL for accessing the Web application
IBM WebSphere	Node Name — Leave blank or specify a node you've set up on your server
Oracle9iAS	<p>Deployment Name — CalculatorWARClient; used in the URL for accessing the Web application</p> <p>Target Path — Leave blank or specify a path you've set up on your server</p> <p>Website Name — Accept the default value or specify a name you've set up on your server</p>



For more details, select the section for your server in the deployment instructions.

- Click **Deploy** in the Deployment Settings dialog.

OR

Click **OK** in Deployment Settings and select **Project>Deploy Archive** from the menu.

Workbench displays progress messages, errors, and warnings on the Deploy tab of the Output Pane.

TIP For most server types, full deployment will fail if your server is not running. For some servers you need to restart after deployment. For details, see the section for your server in the deployment instructions.



EXERCISE 4-11: Test the Calculator Client application

1. Open your browser and enter the URL for the application. It will generally include:

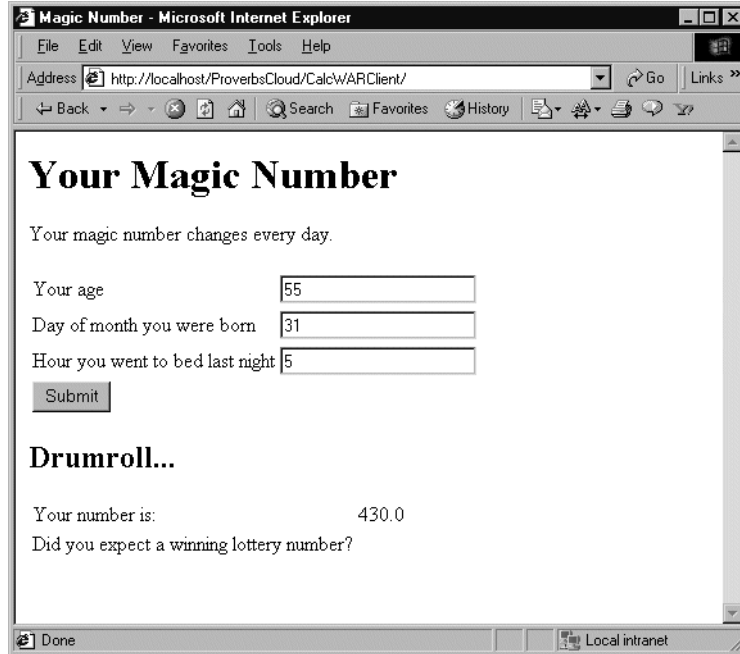
Part of URL	Description	Typical value
Server	URL for the server, including the port number (if not the default port 80) and any server-specific data TIP For a SilverStream server, include the database to which you deployed the WAR	<code>http://localhost/ProverbsCloud/</code> <code>http://www.mydomain.com:8080/</code>
Web application	URL for the WAR TIP For a SilverStream server, this is a relative URL that you specify in the deployment plan	<code>CalcWARClient/</code>
Page	(Optional) URL for the page you want to view; if blank the application displays the welcome page specified in the deployment descriptor	(blank)

For example, if the application is deployed to a local SilverStream server in a database called ProverbsCloud and the URL for the application is CalcWARClient, the URL would be:

```
http://localhost/ProverbsCloud/CalcWARClient
```

You see the welcome page with a form for specifying the calculation data.

2. Enter some values (integers only) in the form and click **Submit**.
The results, displayed by calcnumber.jsp, appear below the form.



Summary of what you've done

Developing the application In this lesson you built a Web application that displayed a form to users, used the form data when it called the Calculator Web Service, and presented the results of the calculation on the same JSP page.

Using Workbench tools You used these tools in Workbench:

- New Project Wizard (File>New Project)
- Project Settings dialog (Project>Project Settings)
- Web Service Wizard (File>New, Web Services tab)
- JSP Wizard and Editor
- Java Class Wizard and Editor
- Deployment tools (Open Deployment Descriptor on project popup menu, Edit>Profiles, Project>Deployment Settings, Project>Deploy Archive)

Next lesson In the next lesson you will learn about additional tools for testing Web Services.

5 Testing Techniques

What you will learn

This lesson teaches you how to use tools for testing your Web Service. You will learn about:

- Viewing the WSDL in your browser
- Inspecting message traffic with TcpTunnel

You'll use the project for the Calculator Web Service you developed in Lesson 2, "Creating a Web Service".

What you will do

1. View the WSDL for the deployed Web Service
2. Edit the client code to redirect messages to TcpTunnel
3. Run the client and observe the message traffic with TcpTunnel

How long will it take? About 10 minutes

NOTE You need to run your J2EE application server to query the Calculator Web Service you deployed in Lesson 2, "Creating a Web Service".

Viewing the WSDL in your browser

As you've seen, the Web Service Wizard adds several Java classes to your project. In addition, the wizard's **Generate WSDL file** option adds a WSDL file to the project. The WSDL file describes your Web Service for clients that don't have access to the actual Web Service code. In Source layout it's in the src directory, and in Archive layout it's in the WEB-INF/classes directory.

If you type the URL for the Web Service in your browser, the jBroker Web code on the server gets a plain GET request, not a SOAP message. So instead of running a Web Service method and returning a SOAP message, it displays the WSDL for the Web Service. With this feature, you can use the Web to give other developers the information they need to develop a client application that calls your Web Service.

NOTE Another way of sharing information about a deployed Web Service is in a registry, described in Lesson 1, "Registries and WSDL for Web Services".



EXERCISE 5-1: View the WSDL for the deployed Web Service

This procedure requires a browser that understands and displays XML, such as Internet Explorer 5 and later.

1. If the application server where you deployed the Calculator Web Service isn't running, start it now.
2. Open your Internet Explorer browser.
3. In Lesson 2, "Creating a Web Service" you specified an URL for the Web Service binding—for example, **http://localhost/ProverbsCloud/Calculator/CalculatorImpl**. Go to that URL in the browser.

The browser displays the WSDL for the Web Service.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <definitions name="CalculatorImplWSService"
  targetNamespace="urn:calc.CalculatorImpl"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:tns="urn:calc.CalculatorImpl"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <types />
  - <message name="addRequest">
    <part name="arg0" type="xsd:double" />
    <part name="arg1" type="xsd:double" />
  </message>
  - <message name="addResponse">
    <part name="result" type="xsd:double" />
  </message>
  - <message name="subtractRequest">
    <part name="arg0" type="xsd:double" />
    <part name="arg1" type="xsd:double" />
  </message>
  - <message name="subtractResponse">
```


Inspecting message traffic with TcpTunnel

One of the tools in jBroker Web is TcpTunnel, a console that displays SOAP request and response messages and HTTP headers sent between a client and a Web Service. The basic steps for using TcpTunnel are:

1. **Alter the binding URL** for the Web Service in the client code to redirect requests to TcpTunnel using localhost and a unique port number.
2. **Start TcpTunnel** with arguments specifying the new port number and the original server name and port number.
3. **Run the client program** and observe the messages on the TcpTunnel console.

These exercises provide detailed steps for running the Calculator client with TcpTunnel. The steps show you how to include the altered binding URL in the test client's code; you could also change the client code to accept the URL as a command-line argument.



EXERCISE 5-2: Edit the client code to redirect messages to TcpTunnel

In this exercise you'll change the binding URL in the client to redirect message traffic through TcpTunnel.

1. Start Workbench and open the Calculator project in the CalculatorWS directory.
TIP If you opened that project recently, you can use the **File>Recent Files** menu item.
2. Open **CalculatorImplWS_Stub.java** in the editor. Find and highlight the binding URL and copy it to the clipboard. The URL is the second string in a line that looks like this:


```
new com.sssw.jbroker.web.Binding("soap",
    "http://localhost/ProverbsCloud/Calculator/CalculatorImpl"),
```
3. Close the file.
4. Open **CalculatorImplWSClient.java** in the editor.
5. Edit the **getRemote()** method to include code for setting the binding, then paste in the URL from the stub:

```
public CalculatorImplWS getRemote(String[] args) throws Exception
{
    InitialContext ctx = new InitialContext();

    String lookup = "xmlrpc:soap:calc.CalculatorImplWSService";
    CalculatorImplWSService service = (CalculatorImplWSService)ctx.lookup(lookup);
    CalculatorImplWS remote = (CalculatorImplWS)service.getCalculatorImplWSPort();

    ((javax.xml.rpc.Stub)remote)._setProperty("javax.xml.rpc.service.endpoint.address",
        "http://localhost/ProverbsCloud/Calculator/CalculatorImpl");
}
```

```
    return remote;  
}
```

6. In the pasted URL, change the server and port to **localhost:9090** but keep the rest of the Web Service's real URL. The port 9090 is an arbitrary unused port number.

The resulting line of code looks like this:

```
((javax.xml.rpc.Stub) remote) ._setProperty("javax.xml.rpc.service.endpoint.address",  
    "http://localhost:9090/ProverbsCloud/Calculator/CalculatorImpl");
```

7. Save the file and close it.
8. Select **Project>Build** to recompile CalculatorImplWSCClient.



EXERCISE 5-3: Run the client and observe the message traffic with TcpTunnel

In this exercise you'll start TcpTunnel and run the test client with the Client Runner window.

1. Start TcpTunnel by opening a DOS window in the **Workbench-install-directory\bin\win32** directory and typing a command in this format:

```
tcptunnel 9090 server-with-deployed-web-service port
```

For example, if the Web Service was deployed to localhost:80, type:

```
tcptunnel 9090 localhost 80
```

If the service was deployed to www.myweb.com, type:

```
tcptunnel 9090 www.myweb.com 80
```

2. In Workbench, run the test client the same way you did in Lesson 2, "Creating a Web Service": select **Project>Run Web Service Client Class** from the menu, select the CalculatorImplWSCClient class, enter two numbers as arguments, and click **Run**.
3. Look at the TcpTunnel console window to see the HTTP headers and SOAP messages being exchanged.

The left pane contains the SOAP requests made by the client, and the right pane displays the responses from the Web Service.

Summary of what you've done

Developing the application In this lesson you found out how to display WSDL for a Web Service published using jBroker Web and you learned how to examine the SOAP message traffic using TcpTunnel.

Using Workbench tools You used these tools in Workbench:

- Edit Pane
- TcpTunnel (jBroker Web command-line tool)

What's next Congratulations. You've finished building the Calculator Web Service and a client Web application for it.

To learn more about J2EE and Workbench, try the WAR tutorial.

Index

A

- archives
 - JAR project (tutorial) 50
 - WAR project (tutorial) 25, 62

C

- Calculator Web Service (tutorial)
 - creating 23
 - deploying 41
 - deployment descriptor 38
 - generating client code 49
 - running test client 45, 56
- CalculatorClient application (tutorial)
 - about 61
 - deploying 81
 - deployment descriptor 78
 - JavaServer Pages for user interface 67
 - testing 84
 - Web Service client code 66

J

- Java class
 - creating (tutorial) 73
- JavaServer Pages
 - for Web Service client (tutorial) 67
 - JavaBean for form (tutorial) 67
- JAX-RPC
 - support for (tutorial) 24
- JBroker Web
 - adding libraries to project (tutorial) 28
 - defined (tutorial) 24

N

- Navigation Pane
 - Registry tab (tutorial) 2

P

- projects
 - creating (tutorial) 25
 - JAR (tutorial) 50
 - WAR for Web Service client (tutorial) 62

T

- TcpTunnel
 - testing Web Service (tutorial) 89
- tutorials
 - developing a Web Service 23
 - Web Service Wizard, client code 49

U

- URLs
 - Web Service (tutorial) 31

W

- Web applications
 - Web Service client (tutorial) 61
- Web Service Wizard
 - generating client code (tutorial) 49, 53, 66
 - generating from source object (tutorial) 31
 - list of generated files (tutorial) 35, 56
- Web Services
 - client WAR application (tutorial) 61
 - creating (tutorial) 23
 - generating client code (tutorial) 49
 - JAX-RPC support (tutorial) 24
 - project classpath (tutorial) 28
 - registries (tutorial) 1
 - registries, about publishing (tutorial) 21
 - registries, business information (tutorial) 5
 - registries, service information (tutorial) 6
 - registry profiles (tutorial) 2
 - RMI model (tutorial) 24
 - skeleton, tie, stub classes (tutorial) 24

- testing message traffic (tutorial) 89
- UDDI, defined (tutorial) 2
- Wizard, available in WSDL Editor (tutorial) 18
- WSDL, about (tutorial) 50
- WSDL, creating client from (tutorial) 49
- WSDL, getting from Web Service (tutorial) 87
- WSDL Editor
 - about (tutorial) 9
 - inserting elements (tutorial) 11
 - stylized view (tutorial) 15
 - toolbar (tutorial) 17