
SilverStream eXtend Workbench

Tutorial: Developing a Web Application

Version 4.0

June 2002



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 or [<mailto: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 or [<mailto: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 or [<mailto: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 ix

Purpose	ix
Audience	ix
Prerequisites	ix
Lessons	x

PART I CREATING A WEB APPLICATION PROJECT

Lesson 1 Architecture of an MVC Application 3

What you will learn	3
What you will do	3
How the Proverbs application implements MVC and Struts	4
What is MVC?	4
What is Struts?	4
MVC structure of the tutorial application	6
Examining the tutorial application	6
EXERCISE 1-1: Start Workbench and open the ProverbFinal project	7
Implementing the view	8
EXERCISE 1-2: Look at source code for the navigation bar	11
How Struts enables internationalization and localization	13
EXERCISE 1-3: Look at the text resources file	14
How to create a form using Struts tags	15
EXERCISE 1-4: Look at the Struts version of an HTML form	15
Implementing the controller	16
EXERCISE 1-5: Look at the ActionServlet javadoc and the ProverbActionServlet class	16
How Struts handles actions	16
EXERCISE 1-6: Look at the Struts configuration file	18
Implementing the model	19
EXERCISE 1-7: Look at the source code for the TodayAction class	19
Data for the application	19
EXERCISE 1-8: Look at source code for accessing proverb data	20
Summary of what you've done	21

Lesson 2 Setting Up Your Data Source 23

- What you will learn 23
- What you will do 23
- Choices for setting up a data source 24
 - Building your own database 24
 - EXERCISE 2-1: Build a database and import proverb data 24
 - Using the Cloudscape database 25
- Adding the Cloudscape database to SilverStream 25
 - EXERCISE 2-2: Add the Cloudscape database to the SilverStream server 26
- Using the Cloudscape database with BEA WebLogic 28
 - EXERCISE 2-3: Edit the startup file for WebLogic to use Cloudscape 28
 - EXERCISE 2-4: Configure a WebLogic connection pool 29
 - EXERCISE 2-5: Configure a WebLogic data source 30
 - EXERCISE 2-6: Configure the connections for the WebLogic pool 30
- Summary of what you've done 31

Lesson 3 Working with Projects and Archives 33

- What you will learn 33
- What you will do 33
- The relationship between projects and archives 33
 - Where source files reside 34
 - Typical directory structure of an archive 35
- Creating a project 36
 - EXERCISE 3-1: Set up directories for your project 36
 - EXERCISE 3-2: Create a new project 37
- Adding content to the project 39
 - EXERCISE 3-3: Add directories to the project 39
 - EXERCISE 3-4: Add content from elsewhere in the file system 41
- Setting up the project's classpath 44
 - EXERCISE 3-5: Set up a classpath for building the project 44
- Summary of what you've done 45

Lesson 4 Deploying and Testing the Welcome Page 47

- What you will learn 47
- What you will do 47
- Adding new files to the project 48
 - EXERCISE 4-1: Add files to the project 48
- Working with JSP pages 50
 - EXERCISE 4-2: Create a new JSP page 50
 - EXERCISE 4-3: Edit the JSP page 53

Building and archiving	55
EXERCISE 4-4: Compile the Java code and generate the archive file	55
Working with the deployment descriptor	57
EXERCISE 4-5: Begin editing the deployment descriptor	57
EXERCISE 4-6: Add initialization parameters for the servlet	60
EXERCISE 4-7: Add a servlet mapping	62
EXERCISE 4-8: Specify the project's default page	63
EXERCISE 4-9: Add tag libraries	63
EXERCISE 4-10: Rebuild the archive	64
Deploying the project	64
EXERCISE 4-11: Deploy the project	65
Testing the application	69
EXERCISE 4-12: Test the application in the browser	69
Summary of what you've done	70

PART II FORMS AND DATA IN A WEB APPLICATION

Lesson 5 Setting Up Database Access 77

What you will learn	77
What you will do	77
Making the data source available to the application	77
Resource references in the deployment descriptor	78
EXERCISE 5-1: Add a resource reference to the deployment descriptor	78
Identifying the database in the server deployment information	82
EXERCISE 5-2: Identify the database in the server deployment information	82
Getting the data source when the application starts	84
EXERCISE 5-3: Extend the Struts ActionServlet to get the data source during initialization	84
EXERCISE 5-4: Change the class for the application's startup servlet	85
Summary of what you've done	86

Lesson 6 Defining an Action That Displays Data 87

What you will learn	87
What you will do	87
Querying the database	88
EXERCISE 6-1: Retrieve data from the database	88

Struts support for an action	89
Telling the controller about a Struts action	90
EXERCISE 6-2: Define the action in the Struts configuration file	91
Retrieving data in the Action class	92
EXERCISE 6-3: Use an Action class to set up the data for a JSP page	92
Struts tags for displaying data	93
EXERCISE 6-4: Display the retrieved data in a JSP page	93
Deploying and testing data access	94
EXERCISE 6-5: Deploy the application	94
EXERCISE 6-6: Test today.jsp	95
Summary of what you've done	96

Lesson 7 Defining a Form and Results Page 97

What you will learn	97
What you will do	97
Two actions for one form	98
EXERCISE 7-1: Define two actions in the Struts configuration file	98
Setting up the form	99
Using Struts tags to define a form	99
EXERCISE 7-2: Examine the form elements in the JSP page	100
Supporting the form with an ActionForm class	100
EXERCISE 7-3: Examine the SelectForm class	101
Processing for the actions	102
EXERCISE 7-4: Examine the SelectAction class	102
Displaying the retrieved data	102
EXERCISE 7-5: Examine the JSP pages that show the results of the search	103
Deploying and testing the form	104
EXERCISE 7-6: Deploy the application	104
EXERCISE 7-7: Test the Find Proverbs activity	104
Summary of what you've done	105

Lesson 8 Defining a Form for Database Update 107

What you will learn	107
What you will do	107
Configuring actions for contributing a proverb	107
EXERCISE 8-1: Define the contribute actions in the Struts configuration file	108
The classes that support the contribute actions	109
EXERCISE 8-2: Examine the code for the contribute actions	109
Deploying and testing the finished application	110

EXERCISE 8-3: Deploy the application	110
EXERCISE 8-4: Test the contribute action and the rest of the application's activities	110
Summary of what you've done	112

About This Book

Purpose

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

- Workbench projects
- J2EE WARs (Web applications packaged in Web archives)
- J2EE application servers
- Struts open source framework for the Model-View-Controller application architecture
- J2EE techniques for database access

Audience

This tutorial is for developers who want an introduction to Workbench projects or want to learn more about Web applications.

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, JavaServer Pages (JSP), and tag libraries
- Understanding of how browsers, application servers, and databases interact in Web applications
- Relational database knowledge

Software In addition to the Workbench software, you need:

- A J2EE application server for deploying the application
- A DBMS for data storage

If you already have this software, you can deploy the standards-based J2EE WAR to your application server using the 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, which includes the Cloudscape DBMS, from www.silverstream.com/downloads. For the tutorial, all you need is the Lite Edition (J2EE server and Cloudscape).

In the supporting tutorial files, you'll find a Cloudscape database with the application data as well as SQL files for building your own database.

Lessons

This tutorial is divided into two parts: "Creating a Web Application Project" and "Forms and Data in a Web Application".

The lessons in **Part I** teach you the basics of Workbench projects and the architecture of a Struts MVC application.

Lesson		Description
1	Architecture of an MVC Application	Examines the architecture of the Proverbs application and how it uses Struts to implement an MVC (Model-View-Controller) design pattern
2	Setting Up Your Data Source	Describes database choices for the Proverbs application and teaches you how to set up the provided Cloudscape database
3	Working with Projects and Archives	Teaches you how to set up projects for Workbench and build a J2EE archive
4	Deploying and Testing the Welcome Page	Introduces the JSP Wizard and server profiles, then teaches you how to deploy an application

The lessons in **Part II** teach you how to access a database in a J2EE application and how to define forms that display and update that data.

Lesson		Description
5	Setting Up Database Access	Illustrates the code for accessing the data source and teaches you how to add database connection information to the application's configuration files
6	Defining an Action That Displays Data	Teaches you about Struts custom tags for displaying data, Java code for handling a Struts action, and configuration settings for the action; examines the method calls that retrieve the data
7	Defining a Form and Results Page	Teaches you about Struts forms
8	Defining a Form for Database Update	Teaches you about coding and configuring actions that update the database

Part I Creating a Web Application Project

This part teaches you how to use SilverStream eXtend Workbench to develop the sample Web application and how the MVC architecture is used in a Struts application.

The lessons are:

- Lesson 1, “Architecture of an MVC Application”
- Lesson 2, “Setting Up Your Data Source”
- Lesson 3, “Working with Projects and Archives”
- Lesson 4, “Deploying and Testing the Welcome Page”

Lesson 1, “Architecture of an MVC Application” provides a walkthrough of the Proverbs Web application, which uses Struts to implement the Model-View-Controller architecture. In this lesson you see how different parts of the application partition the work. You use Workbench to examine the files in a completed project.

In Lesson 2, “Setting Up Your Data Source” you set up a data source of proverbs using the DBMS of your choice.

In Lesson 3, “Working with Projects and Archives” and Lesson 4, “Deploying and Testing the Welcome Page”, you use Workbench to define the Proverb project, edit files, build the archive, and deploy to the server. After you’ve done these lessons, you should feel ready to begin work on your own projects.

1 Architecture of an MVC Application

What you will learn

This lesson describes the architecture of the Proverbs application and how it uses Struts to implement an MVC (Model-View-Controller) design pattern. In the lesson you will use SilverStream eXtend Workbench to examine some files from the completed project to understand how the application fits together. In later lessons you will use Workbench to develop parts of the Proverbs application.

You will learn about:

- How the Proverbs application implements MVC and Struts
- Implementing the view
- Implementing the controller
- Implementing the model
- Data for the application

NOTE If you understand J2EE design and Struts or just want to dive right in, you can skip this guided tour of the application and begin using Workbench in Lesson 2, “Setting Up Your Data Source”.

What you will do

1. Start Workbench and open the ProverbFinal project
2. Look at source code for the navigation bar
3. Look at the text resources file
4. Look at the Struts version of an HTML form
5. Look at the ActionServlet javadoc and the ProverbActionServlet class
6. Look at the Struts configuration file
7. Look at the source code for the TodayAction class
8. Look at source code for accessing proverb data

How long will it take? About 25 minutes

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

How the Proverbs application implements MVC and Struts

The Proverbs tutorial is a Web application—an application that is packaged as a WAR (Web archive) and deployed to a J2EE application server. Standard features of a WAR include:

- A deployment descriptor in XML format
- JSP pages that are accessible to a browser
- Java classes that are hidden from URL access
- Other files—such as JAR files used by the Java classes, image files used by JSP pages, custom tag libraries, and other XML configuration files

What is MVC?

The Model-View-Controller design pattern prescribes a way of partitioning the application's code to keep the user interface (the view) isolated from the business logic (the model). A controller determines how user requests are routed to pages and what business logic is invoked to process each request.

The combination of JSP pages for the view and servlets for the controller is called Model 2 and is the currently accepted way to implement an MVC architecture in a Web application.

The Proverbs application implements the MVC architecture by separating user interface from business logic and managing the application flow with a controller servlet.



Much has been written about MVC architecture, and there isn't space here to describe the subtleties. For more information, see *J2EE Blueprints on the Sun Microsystems Web site* (<http://java.sun.com/blueprints>).

What is Struts?

Struts is part of the Jakarta Project at the Apache Software Foundation. It is a framework that implements MVC for Web applications. It provides a servlet controller, tag libraries, and form classes that handle information display in JSP pages, and a configuration file that tells the controller what classes to instantiate to process application data.

A typical Struts application includes:

- **JSP pages** with Struts custom tags that display text, create forms for data input, and process collections of data for presentation on the page
- **ActionForm bean classes** that populate forms with data and retain data for future requests
- **Action classes** that set up data for JSP pages and process user input

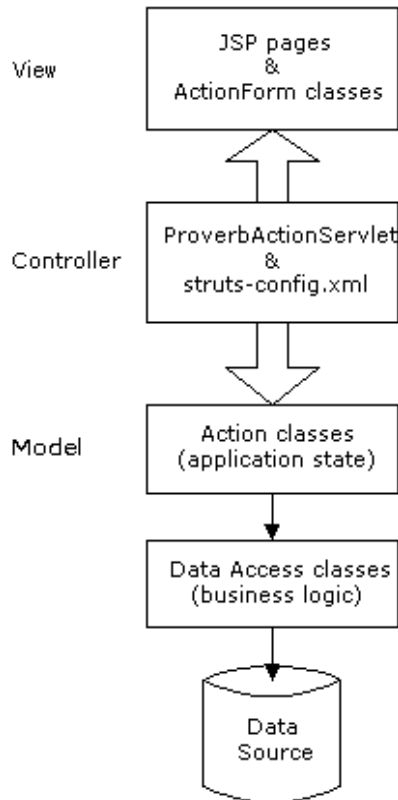
- **An action servlet** that acts as the controller, routing requests to action classes and selecting JSP pages to display
- **A configuration file** that defines the associations between URLs, action classes, form classes, and JSP pages
- **Resource files** that contain the text strings for the application and can be provided in several languages

Here's a quick summary of how Struts implements MVC:

Part	Description
Model	<p>Action classes use the request or the session to store application state information. They can instantiate business logic classes to handle application data.</p> <p>You write an action class for each URL that the controller processes.</p>
View	<p>JSP pages and ActionForm beans display data and forms. ActionForm beans populate form fields with data and retain and validate that data. The data can remain available between requests, and the form can display the previously entered data again.</p> <p>You write an ActionForm class for each form on your JSP pages.</p>
Controller	<p>The ActionServlet class (or your extension of it) runs as a server process and processes URLs it recognizes. It reads the struts-config.xml file to find out what action classes to instantiate and what JSP pages to display for each URL.</p> <p>You can use the ActionServlet class as is or extend it to provide custom behavior.</p>

MVC structure of the tutorial application

The following diagram illustrates how the application implements the MVC pattern.



Examining the tutorial application

To get an overview of the Proverbs application, you'll start Workbench and take a look at the code for the finished version. In later lessons you'll set up a data source and learn how to deploy and run the application.

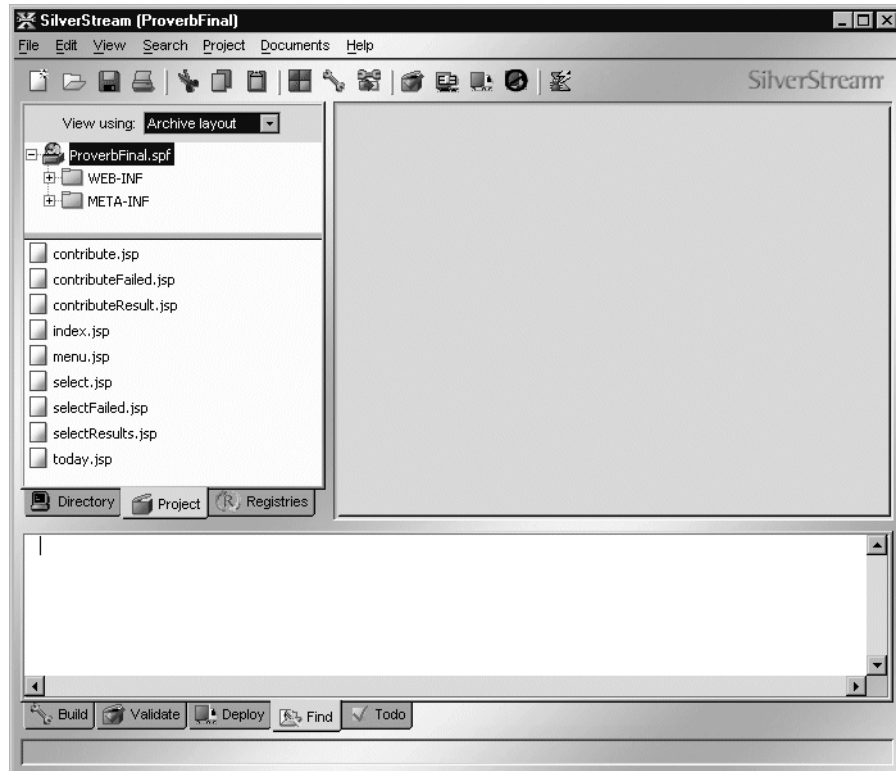


EXERCISE 1-1: Start Workbench and open the ProverbFinal project

1. Start Workbench. You can use the SilverStream Workbench shortcut on the Windows Start menu.
2. Select **File>Open Project**.
3. Find the **ProverbFinal.spf** file in the *Workbench-install-dir/docs/tutorial/ProverbFinal* directory and click **Open**.

NOTE The default installation directory is Program Files/SilverStream/eXtendWorkbench.

4. In the **View using** list box, select **Archive layout** if it isn't already selected.
In the upper left, the Navigation Pane displays the Archive layout of the project.



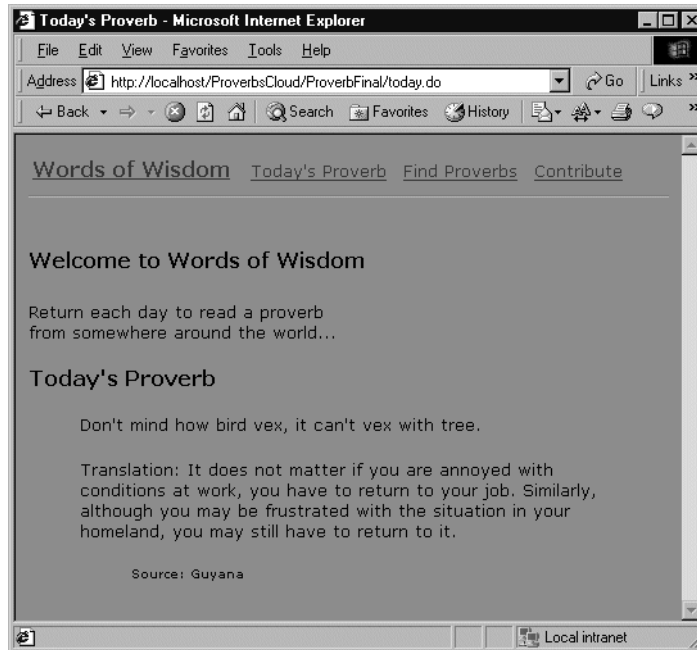
5. Expand the archive folders and look at the files in each folder. The files are displayed in the lower part of the Navigation Pane.

In the rest of this lesson's exercises you will look at MVC and Struts implementation details in these files.

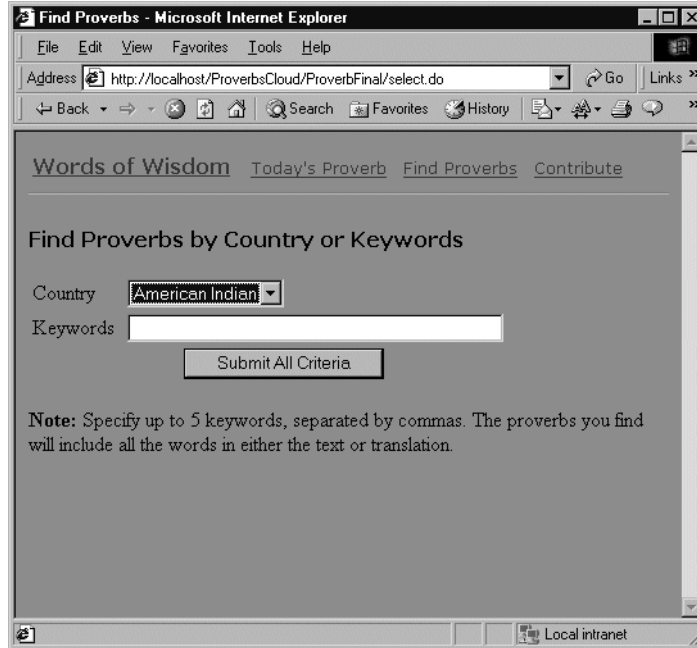
Implementing the view

The Proverbs application has three main activities:

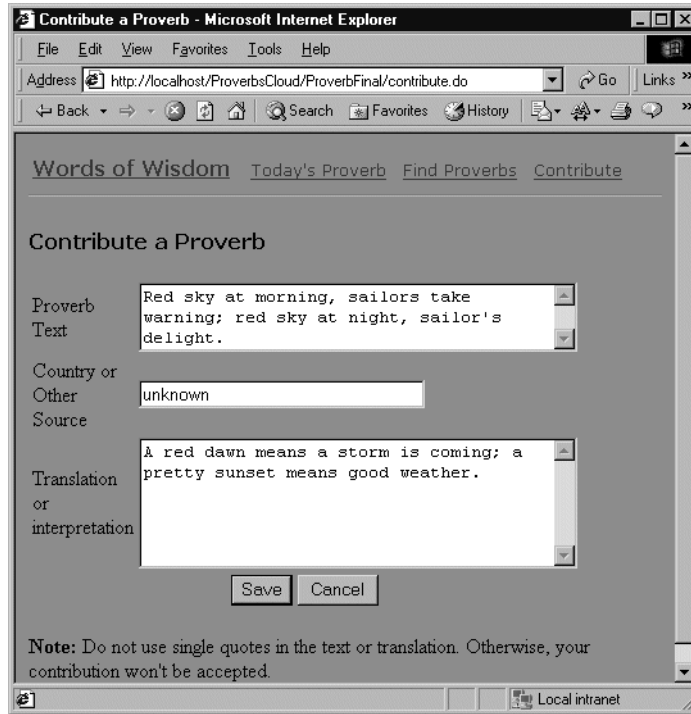
- Displaying the proverb of the day (today.jsp)



- Letting the user search for proverbs by specifying a country that proverbs come from or keywords that the text or translation contains (select.jsp, selectResults.jsp, selectFailed.jsp)



- Letting the user contribute a proverb (contribute.jsp, contributeResult.jsp, contributeFailed.jsp)



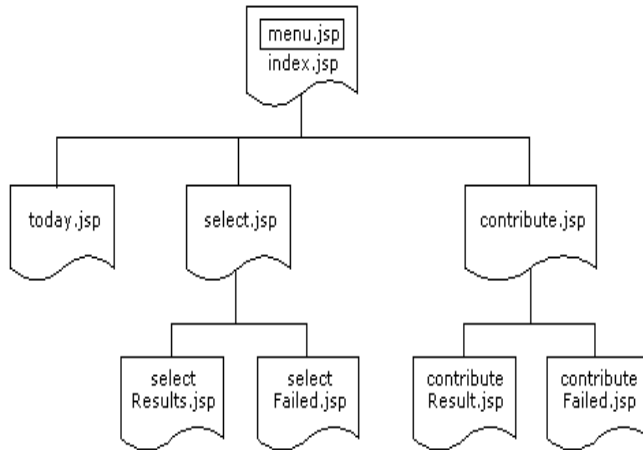
The screenshot shows a Microsoft Internet Explorer window titled "Contribute a Proverb - Microsoft Internet Explorer". The address bar displays "http://localhost/ProverbsCloud/ProverbFinal/contribute.do". The page content includes a navigation menu with "Words of Wisdom", "Today's Proverb", "Find Proverbs", and "Contribute". The main heading is "Contribute a Proverb". The form contains the following fields:

- Proverb Text:** A text area containing "Red sky at morning, sailors take warning; red sky at night, sailor's delight."
- Country or Other Source:** A text input field containing "unknown".
- Translation or interpretation:** A text area containing "A red dawn means a storm is coming; a pretty sunset means good weather."

At the bottom of the form are "Save" and "Cancel" buttons. A note at the bottom of the page reads: "Note: Do not use single quotes in the text or translation. Otherwise, your contribution won't be accepted." The status bar at the bottom indicates "Local intranet".

The welcome or starting page is index.jsp. Navigation is handled by including the JSP fragment menu.jsp in all the other pages. Menu.jsp has links to the three main activities.

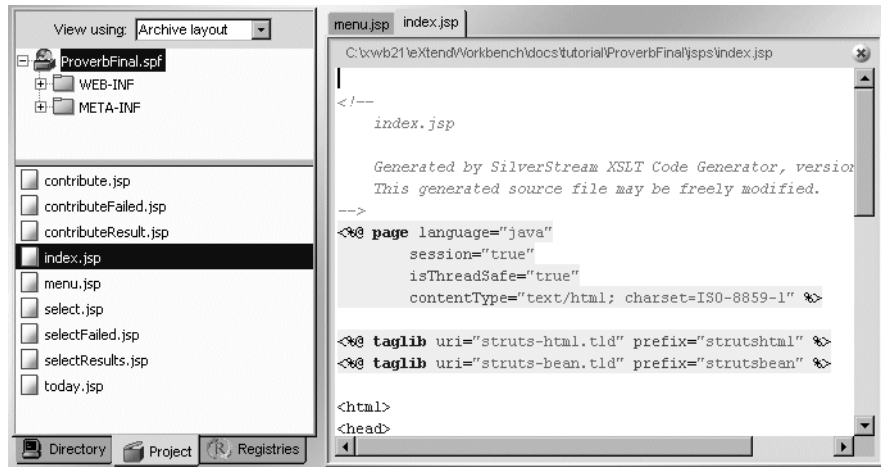
In the diagram you see the three main activities in the second row. If the activity involves submitting a form, the third row shows the pages that display the result.



EXERCISE 1-2: Look at source code for the navigation bar

1. In the Navigation Pane, highlight the project file name **ProverbFinal.spf**.
The JSP pages are at the root of the archive and you see them listed in the lower part of the pane.
2. Double-click **menu.jsp** to open it in the Edit Pane.

4. Double-click **index.jsp**. It opens in the Edit Pane.



5. Notice the taglib directives near the beginning of the file.

```

<%@ taglib uri="struts-html.tld" prefix="strutshtml" %>
<%@ taglib uri="struts-bean.tld" prefix="strutsbean" %>

```

These tell the JSP page where to find the definitions of the custom tags.

6. Notice the JSP include directive after the body tag that inserts the menu code.

```

<%@ include file="menu.jsp"%>

```

The menu code is embedded and then compiled with the rest of index.jsp. There are taglib directives in index.jsp for all libraries used in both index.jsp and menu.jsp.

7. Close each file by clicking its Close button at the upper right in the Edit Pane.

How Struts enables internationalization and localization

Struts uses Java resource bundles and the message custom tag to make text management and localization easier. Although the message tags make it harder to judge the appearance of the JSP pages as you edit, it's easier to edit and translate the application text when it is collected in a single file.

A **resource bundle** is a group of related property files. The main file contains text in the default language for the application. In the file, each property has the following format, where *key* is the value you refer to in the code and *value* is the text the application displays:

```


key=value

```

In the deployment descriptor (`web.xml`), a servlet initialization parameter identifies the properties file that contains the application text.

```
<init-param>
  <param-name>application</param-name>
  <param-value>com.proverb.ApplicationResources</param-value>
</init-param>
```

To provide text in other languages, you create additional files with the same message keys and translated text. Each file includes its two-letter ISO language code in the name, like this: `ApplicationResource_xx.properties`.

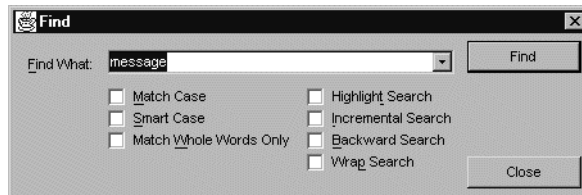
 For more information, see the `ResourceBundle` class in your JDK documentation (java.sun.com/j2se/1.3/docs/api).



EXERCISE 1-3: Look at the text resources file

1. In the Workbench Navigation Pane, expand the `WEB-INF` directory tree and select the `WEB-INF/classes/com/proverb` directory.
2. Find **ApplicationResources.properties** at the end of the list and double-click it to open it in the Edit Pane.
3. Look at the message keys and values. These message keys are used on the welcome page:

```
site.title=Words of Wisdom
site.greeting=Welcome to Words of Wisdom
site.info=Return each day to read a proverb<br>from somewhere around the
world...
```
4. In the Navigation Pane, highlight **ProverbFinal.spf**; then double-click **index.jsp** in the file list.
5. Look for the Struts message tags that refer to the message keys above. Select **Search>Find** and **Find Next (F3)** to help find them all.



6. (Optional) Look for the message tags in the other JSP files too.
7. Close the files.

How to create a form using Struts tags

Custom tags in the Struts HTML tag library implement standard HTML elements and provide hooks to Struts processing. The form tag specifies the target URL for the submitted form and an ActionForm class that has properties for the form's fields. Within each form tag, other Struts tags for the various HTML input fields provide setup and processing.

The action for this form tag is recognized by the controller servlet:

```
<strutshtml:form action="results.do" name="selectForm"
type="com.proverb.SelectForm">
```

For each form, you define an ActionForm class with properties that match the input fields. This class can validate user input and store the data between requests. If the application needs to redisplay the form, the user's data can be restored from this class.



EXERCISE 1-4: Look at the Struts version of an HTML form

1. In the Navigation Pane, highlight **ProverbFinal.spf** so that you see the list of JSP files.
2. Double-click **select.jsp** to open it in the Edit Pane.
3. Find the form tag. It begins **strutshtml:form**.
4. Look at the fields in the form. They are organized in table rows.
 - The country row has a label (strutsbean:message) and a dropdown list (strutshtml:select). An options tag provides the list choices from the countryList property, a Java Collection that is retrieved from the database.
 - The keyword row has a label (strutsbean:message) and an input field (strutshtml:text).
5. (Optional) Open the **ApplicationResources.properties** file again to look up the text that is displayed by each message tag.
6. In the Navigation Pane, expand the **WEB-INF** directory down to the proverb directory and highlight the **proverb** folder.
7. Double-click **SelectForm.java** to open it in the Edit Pane.
8. Notice that it extends ActionForm.

The class is a JavaBean, and the instance variables are properties with get and set methods. The properties correspond with fields on the form. The bean also sets up the countryList Collection as a property. The getList() method looks up the list data in the database.
9. Close the files.

Implementing the controller

The controller is a servlet that listens for URLs and routes the application to the appropriate next step. In the deployment descriptor of the WAR, the servlet mappings specify the custom URLs the servlet will process:

```
<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

In the Proverbs application, the forms use URLs like **contribute.do** and **select.do** to identify the preprocessing and data setup for an associated JSP page. The Struts configuration file (described later) sets up the association for the controller.



EXERCISE 1-5: Look at the `ActionServlet` javadoc and the `ProverbActionServlet` class

1. Open another browser window and visit the Struts Web site at jakarta.apache.org/struts.
2. Click a **Javadoc** link and look at the `ActionServlet` class. The description discusses Struts's MVC implementation.
3. In the Workbench Navigation Pane, expand the **WEB-INF** directory down to the proverb directory and highlight the **proverb** folder.
4. Double-click **ProverbActionServlet.java** to open it in the Edit Pane. Notice that it extends the Struts `ActionServlet` class.
5. Take a look at the `init()` method.

The Proverbs application overrides the `init()` method to do some initialization tasks. It gets a `DataSource` object for the Proverbs database from the server and stores it in the servlet context.

6. Close the file.
7. (Optional) Close the extra browser window.

How Struts handles actions

The `struts-config.xml` file identifies the URLs for the application and describes how the controller processes them.

In the first section of the file, **form-bean** elements identify the classes for the forms in your application:

```
<form-beans>
  <form-bean      name="selectForm"
                  type="com.proverb.SelectForm"/>
```

In the second section, action elements identify the URLs and how they are processed. There is an action element for each of the URLs in your application. Here's the XML for one:

```
<action  path="/select"
         type="com.proverb.SelectAction"
         name="selectForm"
         scope="session"
         validate="false">
  <forward name="success"  path="/select.jsp"/>
  <forward name="failure"  path="/select.jsp"/>
  <forward name="cancel"   path="/index.jsp"/>
</action>
```

A table like this is helpful in planning the application flow and the classes needed at each step:

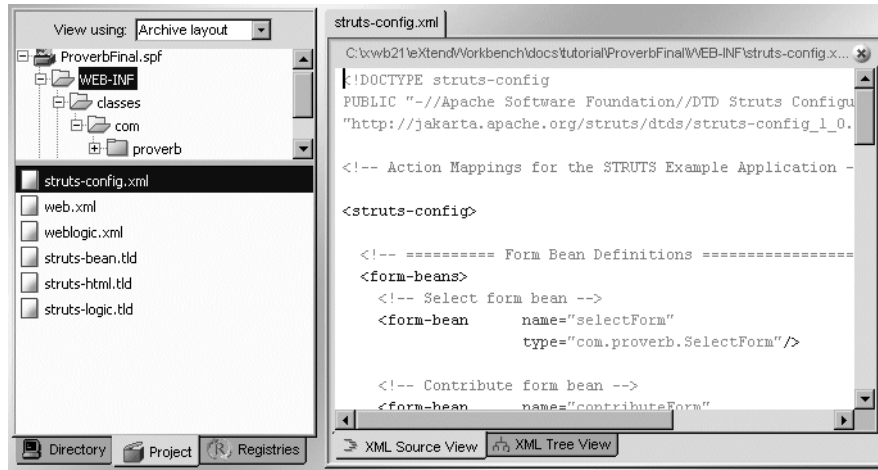
Request URI	Action class (called type)	Form name	Input JSP *	Other attributes	Forward mappings: name & path
select.do	com.proverb.SelectAction	selectForm	—	scope= session validate= false	success; /select.jsp failure; /select.jsp cancel; /index.jsp
results.do	com.proverb.SelectAction	selectForm	select.jsp	scope= session	success; /selectResults.jsp failure; /selectFailed.jsp cancel; /index.jsp

NOTE * The input JSP page is used when the URL is processing a submitted form. For actions that set up the form, the input attribute is omitted.

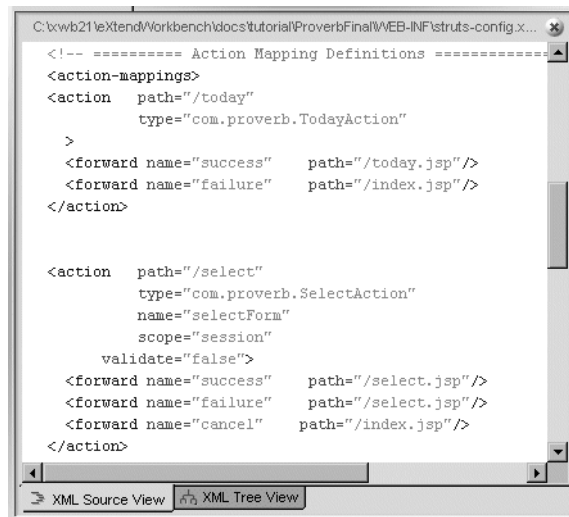


EXERCISE 1-6: Look at the Struts configuration file

1. Highlight **WEB-INF** in the Archive layout of the Navigation Pane.
2. In the file list, double-click **struts-config.xml** to open it in the Edit Pane.



3. Look at the two **form-bean** elements in the form-beans section. They identify the classes that support the forms in the application.
4. Look at the **action** elements in the action-mappings section.



5. (Optional) Study each **action** element and fill out a table like the one above for the rest of the actions. Although a rote activity, it will help you recognize the parts of the application as you encounter them in the rest of the tutorial.
6. When you're done, close the file.

Implementing the model

In a Struts application, the model is implemented by Action classes, which the controller invokes. The Action class changes the application state based on submitted data and other relevant conditions. Action classes can handle all the processing, or they can call other classes for application-specific business logic. In the Proverbs application, the Action classes instantiate other classes to access the database.



EXERCISE 1-7: Look at the source code for the TodayAction class

1. In the Navigation Pane, expand the **WEB-INF** directory, if necessary, and highlight the **proverb** folder.
2. Double-click **TodayAction.java** to open it in the Edit Pane.
3. Look at the code that gets the DataSource that was saved in the servlet context. The code instantiates the ProverbDataAccess class with the DataSource and calls its getTodaysProverb() method.
4. Close the file.

There's more about data access in the next section.

Data for the application

The data for the application is a database of proverbs. The application uses SQL to retrieve the data and update the database when the user contributes a proverb.

- There are only two tables: **proverbs** and **todaysproverb**.
- The columns of the proverbs table are **proverbid**, **proverbtext**, **pvbtranslation**, and **source**.
- The columns of the todaysproverb table are **proverbid** and **dayofyear**.

In a later lesson you will set up a database using the DBMS of your choice. When the data is set up, you can deploy and run the application.



EXERCISE 1-8: Look at source code for accessing proverb data

In this exercise you will look at three files: Constants.java, Proverb.java, and ProverbDataAccess.java.

1. Expand the **WEB-INF** folder and highlight the **proverb** folder.
2. In the file list, double-click **Constants.java** to open it in the Edit Pane.
3. Notice the SQL strings for selecting data and building WHERE clauses. For example, in the data access code, these two strings are joined with **where** to build a query.

```
public static final String SQL_SELECT_PROVERB =
    "SELECT proverbtext, pvbtranslation, source from proverbs ";
public static final String SQL_WHERE_TODAYSPROVERB =
    "proverbid = (select proverbid from todaysproverb where dayofyear=%p)
    ";
```

These constants can be tweaked if necessary for a different DBMS.

4. In the file list, double-click **Proverb.java**.
5. Look at the instance variables that match columns in the Proverb table in the database. A Proverb bean encapsulates a row of data from the database. Each Proverb bean is added to a collection that a JSP page accesses when it displays proverb information.
6. In the file list, double-click **ProverbDataAccess.java**.
7. Notice that the constructor has saved a DataSource object in an instance variable. The caller gets it from the servlet context before instantiating ProverbDataAccess. Do you remember that ProverbActionServlet stored it there during initialization?
8. Select **Search>Find** to look for the method **getProverbList()** or **getTodaysProverb()**. The first half of each of these methods builds a SQL statement. In the second half, the code uses the data source to get a connection. Then it submits the SQL and processes the result set.
9. Close the ProverbFinal project by selecting **File>Close Project** from the menu.

Summary of what you've done

Developing the application In this lesson you learned about the Model-View-Controller design pattern and how the Proverbs application uses Struts to implement MVC.

Using Workbench tools You used these tools in Workbench:

- File>Open Project
- Navigation Pane, Project tab, and Archive layout
- Edit Pane

Next lesson In the next lesson you will learn about the database for the tutorial application.

2 Setting Up Your Data Source

What you will learn

This lesson describes the choices you have for setting up a database for the Proverbs application and shows you how to use the Cloudscape database that is provided with Workbench.

You will learn about:

- Choices for setting up a data source
- Adding the Cloudscape database to SilverStream
- Using the Cloudscape database with BEA WebLogic

What you will do

There are several exercises in this lesson, but you only need to do what applies to your situation:

1. Build a database and import proverb data
You don't need to do this exercise if you can use the provided Cloudscape database with your server.
2. Add the Cloudscape database to the SilverStream server
3. Edit the startup file for WebLogic to use Cloudscape
4. Configure a WebLogic connection pool
5. Configure a WebLogic data source
6. Configure the connections for the WebLogic pool

For other servers, make the database available using your own server tools and procedures.

How long will it take? About 10-25 minutes

NOTE You will need to start your J2EE application server if it needs to be running when you add the database.

Choices for setting up a data source

Since the Proverbs application looks up and stores proverb data, the application needs a data source. Here are two suggestions.

Use the provided Cloudscape database Use the Cloudscape DBMS from IBM and the ProverbsCloud database provided with the tutorial files.

All you need to do is make the database available to your application server. There are instructions for some servers in “Using the Cloudscape database” on page 25. There is also information on where to get Cloudscape.

Build your own database Use another DBMS that is already installed and available for your use. You will need access rights for creating a database with two tables.

If you are building your own database, you can use the provided SQL script for creating the tables and importing the data, described in EXERCISE 2-1: “Build a database and import proverb data” on page 24. The SQL script is correct syntax for the Cloudscape DBMS. You may need to tweak the SQL for your DBMS.

Building your own database

If you are not using the provided ProverbsCloud Cloudscape database, you can use or adapt the SQL script to construct a Proverbs database.



EXERCISE 2-1: Build a database and import proverb data

1. Using your DBMS tools, create an empty database.
2. Locate the **Proverbs.sql** SQL script in the *Workbench-install-dir/docs/tutorial/TutorialFiles/proverbs/sql* directory and open it in a text editor so that you can check the SQL syntax.
The script includes statements for creating two tables and inserting rows into those tables.
3. Edit the SQL for correct syntax, if needed.
4. Use your DBMS tools to import the script. If errors occur, correct the SQL syntax in the editor.
5. Using your application server tools, make the database available to your application server.

Using the Cloudfscape database

The ProverbsCloud database provided with Workbench is already loaded with proverbs from several countries. It also has a table mapping a proverb ID to each day of the year. You'll find the database with the tutorial files in *Workbench-install-dir/docs/tutorial/TutorialFiles/proverbs/dbs*. The following section describes how to make the database available on the SilverStream eXtend Application Server. If you want to use Cloudfscape with another application server, get the Cloudfscape DBMS and use your server tools to make the database available.

About the Cloudfscape DBMS Cloudfscape is a small-footprint embedded DBMS with a free developer's edition. If you already have an application server and want to use the Cloudfscape DBMS, you can download it from www.cloudfscape.com.

The Developer Edition of the SilverStream eXtend Application Server includes the Cloudfscape DBMS. If you need either an application server or a DBMS for this tutorial, you can download an evaluation copy of the SilverStream eXtend Application Server from www.silverstream.com/downloads. For the tutorial, all you need is the Lite Edition (J2EE server and Cloudfscape).

NOTE The ProverbsCloud database was built with **Cloudfscape Version 3.6**. If you have an older version of Cloudfscape, you can either download a newer version or use the SQL script to build a database compatible with your version.

Adding the Cloudfscape database to SilverStream

SilverStream 3.7.2 and later has support for the Cloudfscape DBMS built in. You can use the provided Cloudfscape database with SilverStream Version 3.7.2 and later, including with Version 4.

Using SilverStream Version 4 The original ProverbsCloud database was built with Cloudfscape Version 3.6, which was included with SilverStream 3.7.x. SilverStream Version 4 includes Cloudfscape Version 4. When you add the ProverbsCloud database to the SilverStream Version 4 server using the provided batch file, it is automatically upgraded to Cloudfscape Version 4 (there is an **upgrade** statement at the end of the JDBC URL in the SilverCmd input file you will use to add the database to a Version 4 server).

This means that after adding the database to a Version 4 server, you cannot use the database with a Version 3.7.x server (which doesn't support Cloudfscape Version 4). If you want to use a ProverbsCloud database against both SilverStream versions, you should make copies of the databases—one for each SilverStream version—before adding them to the server.

Checking your environment The batch file that adds the database to the SilverStream server relies on the SILVERSTREAM_HOME environment variable, which is created by the server's installation program. Before proceeding, make sure you have a SILVERSTREAM_HOME environment variable and that it points to the correct version of the SilverStream server.



EXERCISE 2-2: Add the Cloudscape database to the SilverStream server

In this exercise you will use the SilverStream SilverCmd tool to add the ProverbsCloud database to your SilverStream server.

1. Start the SilverStream eXtend Application Server.
2. Open the SilverCmd input file for editing. The file is in the directory *Workbench-install-dir/docs/tutorial/TutorialFiles/proverbs/dbs*.

Server version	File to edit
3.7.x	addProverbsCloud37.xml
4	addProverbsCloud4.xml

If you didn't install the SilverStream server in the default directory, you need to change the path for the SilverStream DTD.

3. Near the beginning of the file, find the DTD name in the DOCTYPE element. It looks like this:

Server version	DTD name
3.7.x	c:/SilverStream37/Resources/DTDCatalog/add_database.dtd
4	C:/Program Files/SilverStream/eXtendAppServer/Resources/DTDCatalog/add_database.dtd

4. Specify the correct path for the DTD by specifying the path for your SilverStream server installation directory.

If you didn't install Workbench in the default directory, you need to change the path for the ProverbsCloud database.

5. Find the `JDBC_URL` element. It looks like this:

Server version	JDBC_URL definition
3.7.x	<code><JDBC_URL>jdbc:cloudscape:c:/Program Files/SilverStream/eXtendWorkbench/docs/tutorial/TutorialFiles/proverbs/dbs/ProverbsCloud</JDBC_URL></code>
4	<code><JDBC_URL>jdbc:cloudscape:c:/Program Files/SilverStream/eXtendWorkbench/docs/tutorial/TutorialFiles/proverbs/dbs/ProverbsCloud;upgrade=true</JDBC_URL></code>

6. Change the default installation directory after **cloudscape**: (c:\Program Files\SilverStream\eXtendWorkbench) to specify your Workbench installation directory.
7. Save **addProverbsCloud37.xml** or **addProverbsCloud4.xml** and close it.
8. Open a command prompt in the *Workbench-install-dir/docs/tutorial/TutorialFiles/proverbs/dbs* directory.
9. For an unsecured server, execute this command:

Server version	Command to execute
3.7.x	<code>addProverbsCloud37 your-server-name</code>
4	<code>addProverbsCloud4 your-server-name</code>

NOTE If your server is running locally, you can specify **localhost** for the server name. For a secured server, specify your user ID and password too. For example, if the user ID is **admin** and the password is **pwd**, type the following command:

Server version	Command to execute
3.7.x	<code>addProverbsCloud37 your-server-name admin pwd</code>
4	<code>addProverbsCloud4 your-server-name admin pwd</code>

The database is now available to the SilverStream eXtend Application Server.

Using the Cloudscape database with BEA WebLogic

These exercises describe how to:

- Configure the WebLogic server to use the Cloudscape DBMS in its embedded form
- Configure a connection pool and data source

NOTE These instructions apply to WebLogic 6.0 SP1 and describe how to configure the default server. Although WebLogic 6.0 SP1 includes Cloudscape 3.5 and an ExamplesServer already configured to run it, you will need to get Cloudscape 3.6 to use the prebuilt ProverbsCloud database.



EXERCISE 2-3: Edit the startup file for WebLogic to use Cloudscape

In this exercise you'll change the classpath and the startup command for WebLogic to use the Cloudscape DBMS in embedded mode.

1. Get Cloudscape 3.6 and run its setup program. You can download it with the SilverStream eXtend Application Server from www.silverstream.com/downloads. For the tutorial all you need is the Lite Edition (J2EE server and Cloudscape). You can also get it from cloudscape.com.
2. Open the startup file for the server in Workbench (or any text editor). It's called **startWebLogic.cmd**.

You'll find it in the configuration directory for the default server. For example, if you installed in the default directory and you called the server mydomain, you'll find it in C:\bea\wlserver6.0sp1\config\mydomain.

3. Change the classpath command to include **cloudscape.jar**. It should look like this, with an appropriate path for cloudscape.jar:

```
set
CLASSPATH=.;.\lib\weblogic_sp.jar;.\lib\weblogic.jar;c:\Cloudscape\lib\cloudscape.jar
```

4. Add the Cloudscape home directory to the server startup command. The home directory is the default location for Cloudscape database directories. The startup command might look like this:

```
"%JAVA_HOME%\bin\java" -hotspot -ms64m -mx64m -classpath %CLASSPATH%
-Dweblogic.Domain=mydomain -Dweblogic.Name=myserver "-Dbea.home=C:\bea"
-Dcloudscape.system.home=c:\Cloudscape\demo\databases
"-Djava.security.policy==C:\bea\wlserver6.0sp1\lib\weblogic.policy"
-Dweblogic.management.password=%WLS_PW% weblogic.Server
```

NOTE There is a space but no carriage return at the end of each line except the last.

5. Save and close the file.

- Start the server by running **startWebLogic.cmd** at a command prompt.



EXERCISE 2-4: Configure a WebLogic connection pool

These instructions describe how to use the WebLogic 6.0 SP1 administration console to configure the database. The steps for other versions of the server may vary.

- With the WebLogic server running, start the administration console in a Web browser. The default URL on a local server is:

```
http://localhost:7001/console
```

In the left panel, the top node is the parent name for your server. Below that you'll see a node called **Services**.

- Expand the **Services>JDBC** node and highlight **Connection Pools**.
- In the right panel, click **Create a new JDBC Connection Pool**.
- Fill in the form with data like this:

Item	Value	Comments
Name	ProverbsCloudPool	An arbitrary name that you'll use again in the DataSource definition
URL	jdbc:cloudscape:c:\Program Files\SilverStream\extendWorkbench\docs\tutorial\TutorialFiles\proverbs\dbs\ProverbsCloud	The format is jdbc:cloudscape:databasepath The URL ends with the directory that hold the database files. The value shown points to the database in the default installation directory for Workbench
Driver Classname	COM.cloudscape.core.JDBCDriver	A class in cloudscape.jar
Properties	user=APP password=password	The user ID and password for the ProverbsCloud database

- Click **Create**.
- Select the **Targets** tab.

7. In the **Available** list, select **myserver** (or the name of your server) and click the right arrow so that it appears in the **Chosen** list. Then click **Apply**.



EXERCISE 2-5: Configure a WebLogic data source

1. With the administration console still running in the browser, highlight **Data Sources** in the left panel under the **Services>JDBC** node.
2. In the right panel, click **Create a new JDBC Data Source**.
3. Fill in the form with data like this:

Item	Value	Comments
Name	ProverbsCloud	An arbitrary name for this definition; for simplicity use the JNDI name
JNDI Name	ProverbsCloud	The JNDI name associated with the resource reference in the weblogic.xml configuration file
Pool Name	ProverbsCloudPool	The name of the connection pool for the database you want to access

4. Click **Create**.
5. Select the **Targets** tab.
6. In the **Available** list, select **myserver** (or the name of your server) and click the right arrow so that it appears in the **Chosen** list. Then click **Apply**.



EXERCISE 2-6: Configure the connections for the WebLogic pool

1. With the administration console still running in the browser, highlight **ProverbsCloudPool** in the left panel under the **Services>JDBC>Connection Pools** node.
2. Select the **Configuration** tab in the top row, then select **Connections** tab in the second row.
3. Change **Initial Capacity** to **1**. Leave the other values as is, including Maximum Capacity of 1. The developer edition of Cloudscape supports only one connection.
4. Restart the server.

The database is ready for use.

Summary of what you've done

Developing the application In this lesson you built or learned about these parts of the Proverbs tutorial application:

- Database built from the provided SQL script
OR
Cloudscape database provided as part of the tutorial
- Application server procedures for using the database

Next lesson In the next lesson you will learn how to set up a WAR project in Workbench.

3 Working with Projects and Archives

What you will learn

In this lesson you will learn about projects for Workbench and how Workbench helps you build a J2EE archive from the files in your project. You will create a project for a Web application that displays proverbs.

You will learn about:

- The relationship between projects and archives
- Creating a project
- Adding content to the project
- Setting up the project's classpath

What you will do

1. Set up directories for your project
2. Create a new project
3. Add directories to the project
4. Add content from elsewhere in the file system
5. Set up a classpath for building the project

How long will it take? About 10 minutes

NOTE You don't need to be running your J2EE application server at this stage in the project's development.

The relationship between projects and archives

In your previous work (before using Workbench), if you used the JAR command to build an archive, you had to arrange the directory structure on your hard disk to mirror the required archive structure. This forced file arrangement isn't necessarily the most convenient way to work.

In Workbench you can group files for different parts of the project in the directories you want. Your project settings specify where those files belong in the archive. Files in different source directories can be assigned to a single directory in the archive.

In general, the content of your project will be directories, not individual files. As you work, you can add files to the project directories and they will be automatically included in the resulting archive. When you specify project content at the directory level, all the files in a source directory will be together in the archive directory you specify. To put files in different archive directories, you should put them in different project source directories.

You can also specify individual files as content for your project. You can give each file a particular location in the archive. To avoid adding content to a project twice, you wouldn't want to add a directory to the project and also add a file in that directory.

Where source files reside

You can organize your source files in many different ways. Code, HTML, and other files for the application you are working on will typically be under the project's root directory. Other files, such as JARs and tag libraries, might be stored in a directory that is used by many projects.

In the Proverbs tutorial application, the project root contains these directories:

Directory	Contents
src	Java source code for the Proverbs application
jsps	JSP pages
resources	Resource bundle file of text strings, which can be translated for internationalization
WEB-INF	Deployment descriptor and Struts configuration file, plus deployment files for specific application servers

Typical directory structure of an archive

The internal directory structure of a J2EE archive is largely up to you. An archive for a Web application (WAR) has only a few requirements. You put files that are not accessible via an URL in the WEB-INF directory and its subdirectories. The deployment descriptor must be in WEB-INF.

In the WEB-INF directory, you can expect to find:

File or directory	Contents
web.xml	The required deployment descriptor that tells the application server how to interact with the Web application
WEB-INF/classes	The compiled Java class files for the application
WEB-INF/lib	JAR files used by the application

The locations of other files are up to you. In the Proverbs application, the Struts tag libraries (TLD files) and configuration file (struts-config.xml) are also in the WEB-INF directory. The JSP pages, which are accessible via URL, are in the root directory of the archive. Files under the WEB-INF directory are not accessible via URL by default, although you can configure them for URL access.

The resulting archive directory structure is:

```
archive root
  jsps
  /WEB-INF
    web.xml
    struts-config.xml
    struts-html.tld, struts-logic.tld, struts-bean.tld
    /classes
      compiled classes in the com.proverb package
    /lib
      struts.jar
```

Creating a project

To create a project for a Web application, first you'll do a little directory setup. Then you'll start Workbench to create the project file.



EXERCISE 3-1: 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 **ProverbStart**. 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 you created ProverbStart in a WorkbenchProjects directory. On Windows, it would look like this:

```
c:\WorkbenchProjects\ProverbStart
```

2. In the ProverbStart directory, create four subdirectories:

```
jsp  
resources  
src  
WEB-INF
```

NOTE WEB-INF must be all uppercase. Windows Explorer might display the uppercase name as Web-inf, but if you type it correctly it will be correct in the project.

3. To speed up the tutorial, most of the Java and JSP code is provided. To include this code, copy these items to the specified directories:

From this directory in <i>Workbench-install-dir/docs/tutorial/TutorialFiles/proverbs/</i>	Copy this	To this project directory
jsp	JSP files	ProverbStart/jsp
src	com folder and its contents	ProverbStart/src
config	struts-config.xml	ProverbStart/WEB-INF



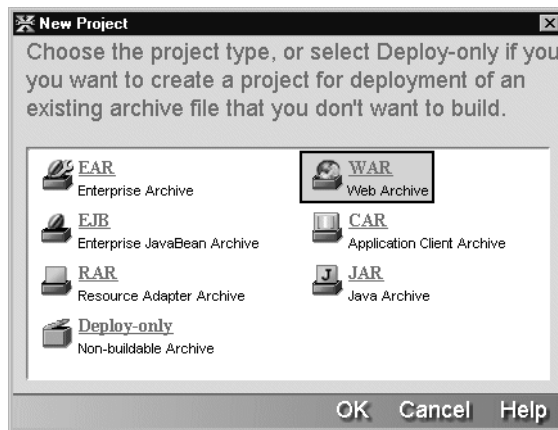
EXERCISE 3-2: Create a new project

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

1. Start Workbench. You can use the SilverStream eXtend Workbench shortcut on the Windows Start menu.

If Workbench is already open, select **File>Close Project** to close the open project, if any.

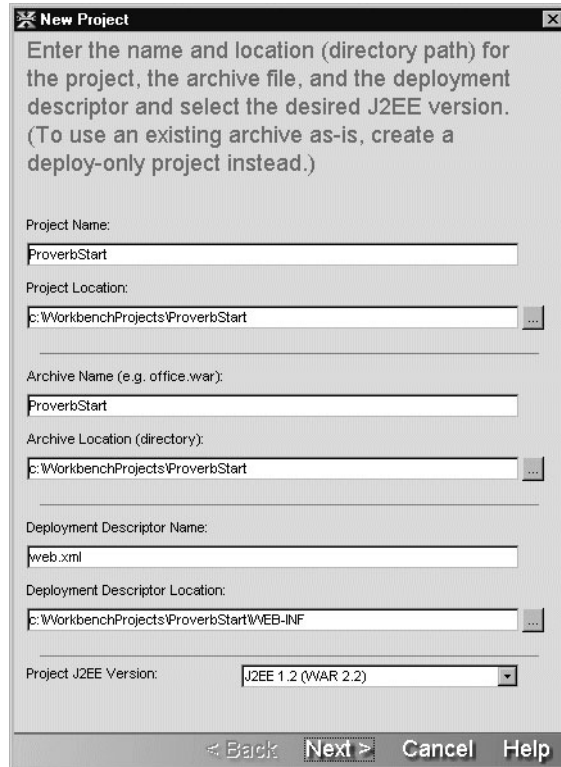
2. Select **File>New Project** from the menu.



3. In the New Project Wizard, select **WAR** and then click **OK**.
4. In the Project Name field, type **ProverbStart**.
5. Click the ellipses beside the Project Location field and select the ProverbStart directory you created in EXERCISE 3-1: “Set up directories for your project”. When you click **OK**, other fields in the dialog are filled in automatically.

The archive location is the project root directory, and the deployment descriptor is in the WEB-INF directory.

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.



7. Click **Next**.

The wizard summarizes the project information.

NOTE If you hadn't closed open projects before you selected New Project, the wizard might ask if you wanted to make the new project a subproject of the open project. This is useful for making a new WAR or EJB part of an EAR.

8. Click **Finish**.

In the Navigation Pane, the Project tab displays the new project. You can use either a Source view or an Archive view.

As you'll see, the contents of the project root directory are not automatically part of the project. You can have other files and directories in the root directory too. In the next exercise you will add the other directories to the project and change the descriptor's entry from a file to a directory.

Adding content to the project

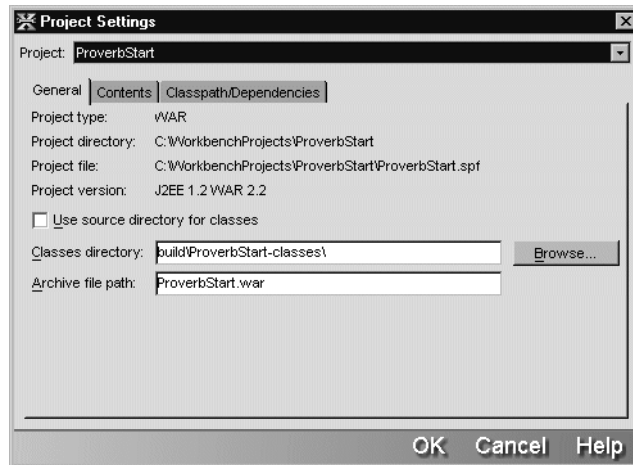
You can add directories and files to a project in two ways: in the Project Settings dialog and from the Navigation Pane. Only the descriptor file, which you specify in the New Project Wizard, is automatically included. The directories and files you select can be in the project root directory or somewhere else in your file system. The next two exercises show you how to add content.



EXERCISE 3-3: Add directories to the project

In this exercise you'll select directories for the project and specify their location in the resulting archive.

1. Select **Project>Project Settings** from the menu.



2. Select the **Contents** tab.

The ProverbStart project is selected in the Project field. If your project included other subprojects, you could choose among the available projects.

3. On the **Contents** tab, highlight **WEB-INF/web.xml** and click **Delete**.

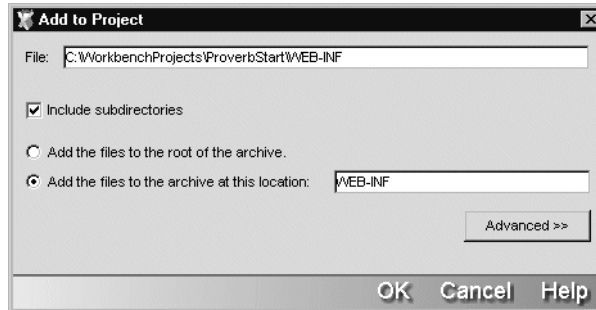
The goal of this step is to create a project entry for the **WEB-INF** *directory* instead of the individual file.

4. Click **Add Directory**.

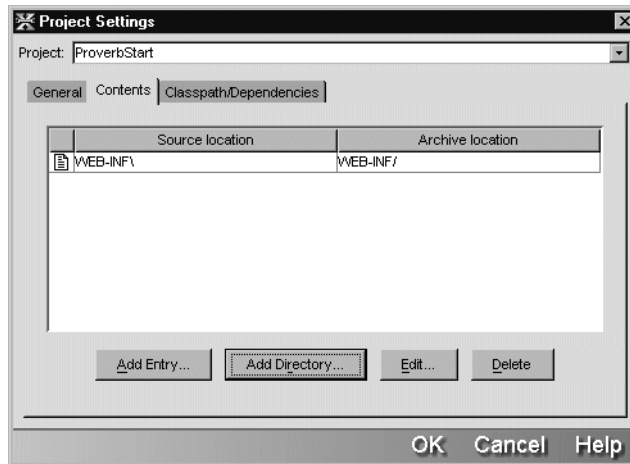
An Open dialog displays the drives and directories on your file system.

5. Navigate the directory structure to find the **WEB-INF** directory in your project root directory and click **OK**.

6. In the Add to Project dialog, select the last item (**Add the files to the archive at this location**) and specify **WEB-INF** for the location.



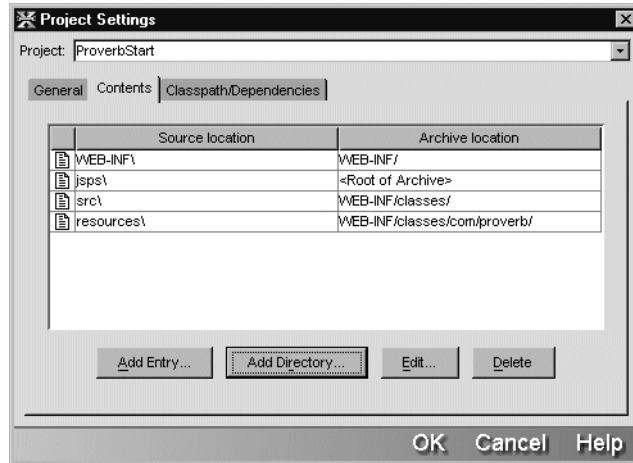
7. Click **OK** to return to the Project Settings dialog.



Instead of the file web.xml, you see the WEB-INF directory as both the source location and the archive location. Later when you add other files to WEB-INF, they will automatically become part of the project.

8. Using the **Add Directory** button, add three more source directories to the project. They are all under the project root directory.
 - Add the **jsps** directory with the option **Add the files to the root of the archive**.
 - Add the **src** directory with the option **Add the files to the archive at this location** and **WEB-INF/classes** as the location.
 - Add the **resources** directory with the option **Add the files to the archive at this location** and **WEB-INF/classes/com/proverb** as the location.

When you're done, the Project Settings should look like this:



9. Click **OK** to close the Project Settings dialog.
10. In the Navigation Pane, switch between Source Layout view and Archive Layout view and expand the directories you see. Notice how Workbench rearranges the source directories to show you the Archive layout.



EXERCISE 3-4: Add content from elsewhere in the file system

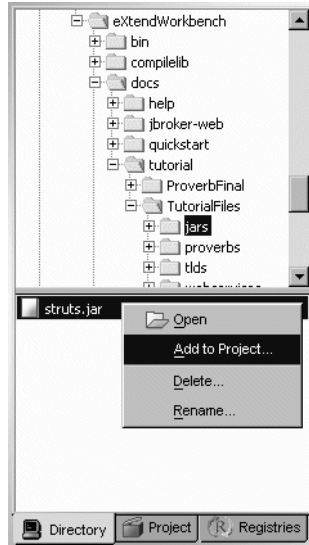
There are other ways to add directories and files to the project besides the Project Settings dialog. In this exercise you will use the Directory tab in the Navigation Pane to add content.

The new files (the Struts JAR and tag libraries) are required in every Struts project. You won't be changing them, so you don't want separate copies in every Struts project you work on. You'll add them directly from the provided TutorialFiles directory.

1. In the Navigation Pane, click the **Directory** tab.

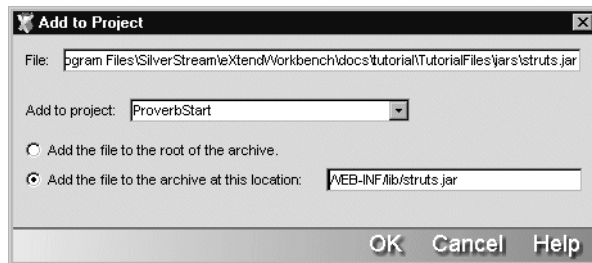
TIP For better viewing, you can resize the Navigation Pane. Move the bottom or right edge to make the whole pane larger or move the bar between the directory and file lists.
2. In the directory list, find the Workbench install directory and expand it to show **Workbench-install-dir/docs/tutorial/TutorialFiles/jars** directory.

NOTE The default installation directory is Program Files/SilverStream/eXtendWorkbench.



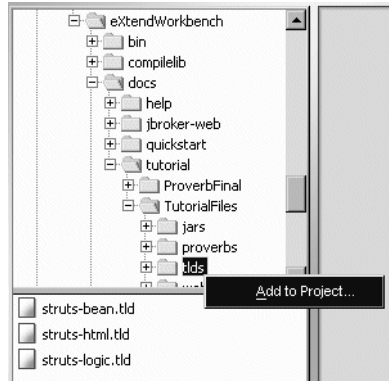
3. In the lower part of the pane, right-click the file **struts.jar** and select **Add to Project**.
4. In the Add to Project dialog, select the option **Add the file to the archive at this location** with a location of **WEB-INF/lib/struts.jar**.

NOTE When you add a file, the location includes the file name.

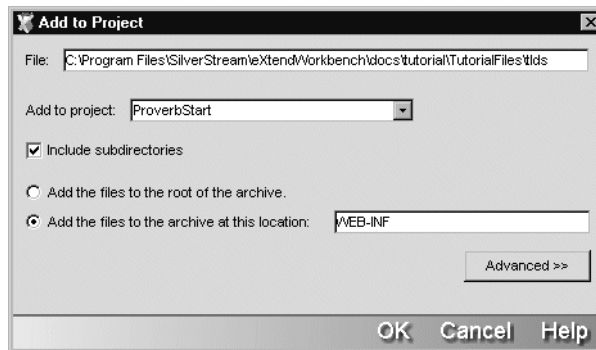


5. Click **OK**.

6. Add the **tlds** directory, which is also in TutorialFiles, by right-clicking the **tlds** directory (not the files this time) and selecting **Add to Project**.



7. In the Add to Project dialog, select the option **Add the file to the archive at this location** with a location of **WEB-INF**.



8. Click **OK**.
9. After closing the Add to Project dialog, select the **Project** tab in the Navigation Pane and select **Archive layout**.

When you click the **WEB-INF** directory, its file list includes **web.xml**, **struts-config.xml**, and the tag libraries (with extension **tld**). When you expand **WEB-INF** and click **lib**, the file list displays **struts.jar**.

NOTE In this exercise you used the Directory tab to select directories and files for your project. The Directory tab displays directories and files and allows you to delete and rename files, but it is not a full-fledged tool for working with your hard disk. To reorganize directories and files, use your operating system's tools.

Setting up the project's classpath

When you build your project, Workbench needs to know where to find JARs and other Java files that your source code refers to. You specify a classpath for building the project in the Project Settings dialog. Workbench automatically includes a JAR of the J2EE classes.

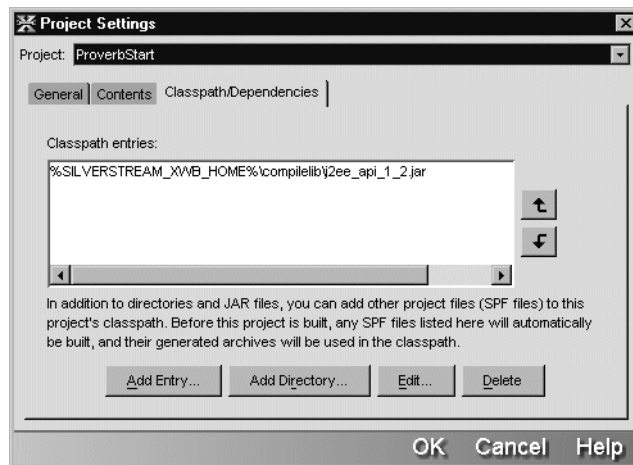


EXERCISE 3-5: Set up a classpath for building the project

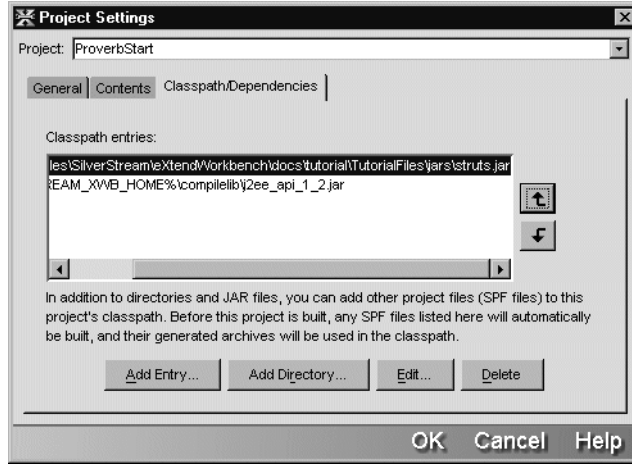
In this exercise you will use the Project Settings dialog to add to the compile-time classpath. For the Proverbs application, the classpath needs to also include struts.jar.

1. With your project open, choose **Project>Project Settings** from the menu.
2. Select the **Classpath/Dependencies** tab.

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



3. Click the **Add Entry** button.
4. Select **struts.jar** in the *Workbench-install-dir/docs/tutorial/TutorialFiles/jars* directory. The list for your classpath should look like this:



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

Summary of what you've done

Developing the application In this lesson you built these parts of the Proverbs tutorial application:

- A source directory structure that included some application files
- A project for the Proverbs Web application
- Archive locations for the directories that are part of the project
- A classpath for building the project

Using Workbench tools You used these tools in Workbench:

- New Project Wizard (File>New Project)
- Source layout and Archive layout on the Project tab of the Navigation Pane
- Project Settings dialog (Project>Project Settings)
- Directory tab in the Navigation Pane and the Add to Project menu item

You can return to the Project Settings dialog whenever you need to make changes in the source directories or archive locations for your project.

Next lesson In the next lesson you will learn about the JSP Wizard and building and deploying projects.

4 Deploying and Testing the Welcome Page

What you will learn

In this lesson you will learn about the JSP Wizard. You will also learn about server profiles and how to use Workbench deployment capabilities.

You will learn about:

- Adding new files to the project
- Working with JSP pages
- Building and archiving
- Working with the deployment descriptor
- Testing the application

What you will do

1. Add files to the project
2. Create a new JSP page
3. Edit the JSP page
4. Compile the Java code and generate the archive file
5. Begin editing the deployment descriptor
6. Add initialization parameters for the servlet
7. Add a servlet mapping
8. Specify the project's default page
9. Add tag libraries
10. Rebuild the archive
11. Deploy the project
12. Test the application in the browser

How long will it take? About 30 minutes

NOTE Your J2EE application server needs to be running for the deployment and testing exercises.

Adding new files to the project

In addition to adding directories that contain files to your project, there are several other ways to add files. You can:

- Move a file into a project source directory
- Save a new file into a project source directory
- Get files from source control, including new files added to project directories by coworkers
- Use the Add to Project menu item to add individual files that are not in project directories

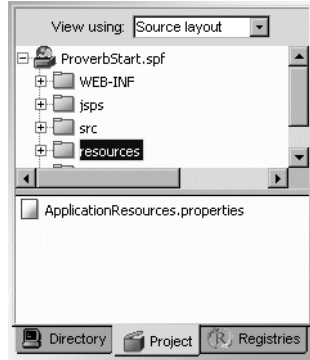


EXERCISE 4-1: Add files to the project

In this exercise you will add a file to the project by copying it from somewhere else and putting it in a directory that is already part of the project.

1. Using your operating system tools, find the file **ApplicationResources.properties** in the *Workbench-install-dir/docs/tutorial/TutorialFiles/proverbs/resources* directory.
2. Copy the file to the **resources** directory under the project root.
You created the resources directory in Lesson 3, “Working with Projects and Archives”. If the project root is *c:/WorkbenchProjects/ProverbStart*, copy the file to *c:/WorkbenchProjects/ProverbStart/resources*.
3. In Workbench on the Project tab of the Navigation Pane, set the view to **Source layout**.
4. Select the **resources** directory.

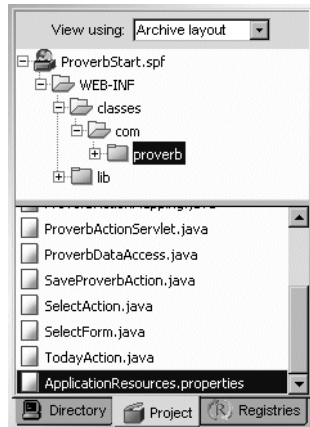
In the file list in the lower half of the pane, you should see the file you just added.



5. Switch to **Archive layout**.
6. Expand the **WEB-INF/classes/com/proverb** directory and highlight **proverb**.

In the file list you should see the `ApplicationResources.properties` file.

In the previous lesson you designated that files in the resources directory belonged with the Java class files in the archive. The Navigation Pane shows you `ApplicationResources.properties` in that location.



Working with JSP pages

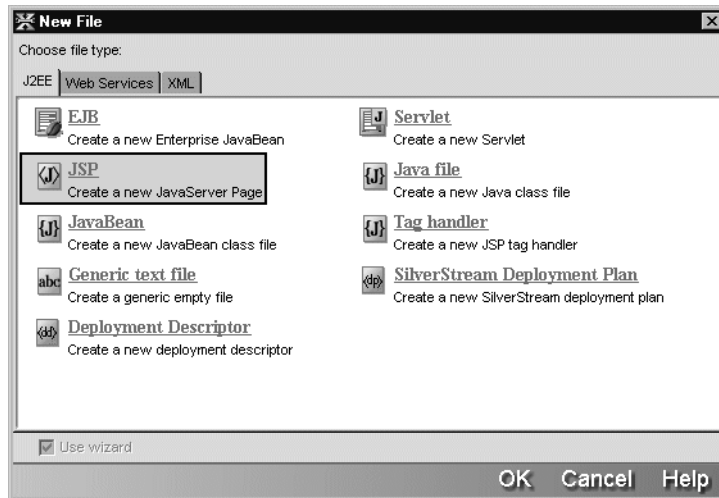
To create new JSP pages, you can use the JSP Wizard, which constructs the skeleton of a JSP file and opens it in the Edit Pane. For existing JSP pages, you can double-click them in the Navigation Pane to open them in the Edit Pane.



EXERCISE 4-2: 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**.

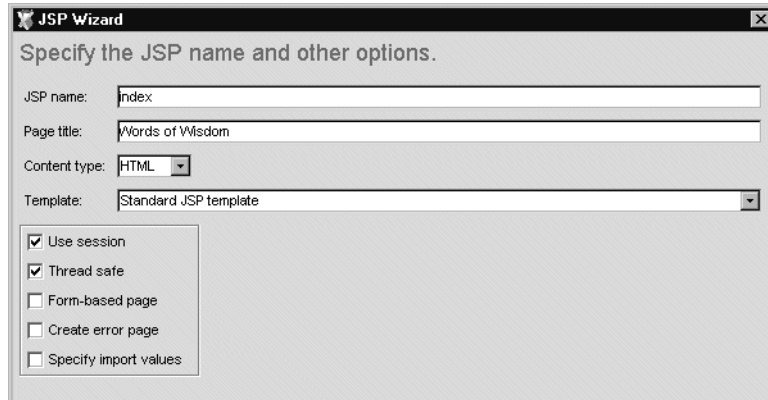


2. In the New File dialog, select **JSP** and click **OK**.
Workbench displays the JSP Wizard.
3. Fill out the first panel of the wizard with this information:

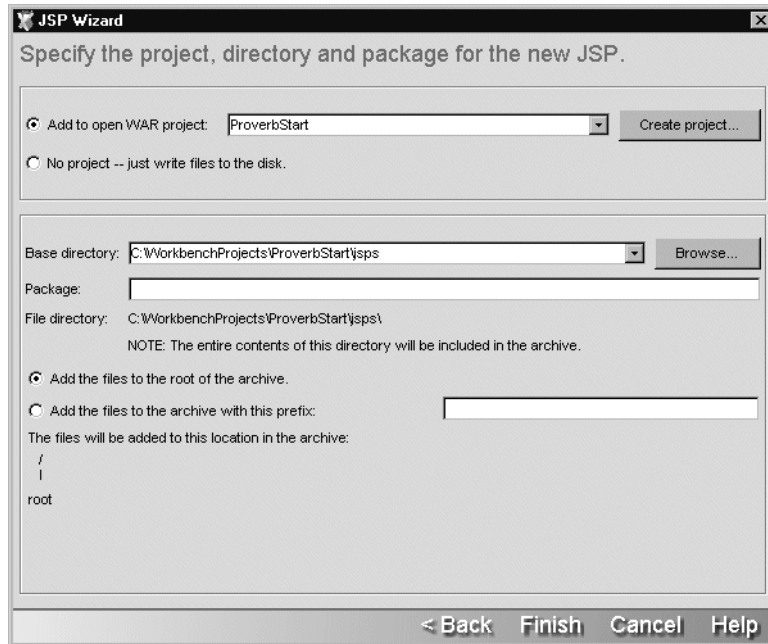
Option	Value
JSP name	index (don't specify the jsp extension)
Page title	Words of Wisdom
Content type	HTML (the default)
Template	Standard JSP template (the default)

4. Select only **Use session** and **Thread safe**.

The first panel of the wizard looks like this:



5. Click **Next**.
6. On the second panel, leave **Add to open WAR project** selected.
7. Specify where to put the file in the project and the archive:
 - For Base directory, select the *project-root\jsp*s directory from the dropdown list—for example, c:\WorkbenchProjects\ProverbStart\jsp
 - 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.



8. Click **Finish**.
9. 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, index.jsp is in the jsps directory in the Source layout and at the archive root in the Archive layout.

More about the wizard When you check **Specify import values** on the first panel of the wizard, it displays a third panel for specifying those values.



EXERCISE 4-3: Edit the JSP page

In this exercise you will change the generated code for `index.jsp` to use Struts tags, get text from the `ApplicationResources` file, and include a navigation menu.

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

OR

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

1. Insert a blank line after the closing bracket (>) of the `@page` directive and add these lines:

```
<%@ taglib uri="struts-html.tld" prefix="strutshtml" %>
<%@ taglib uri="struts-bean.tld" prefix="strutsbean" %>
```

2. Insert a blank line below the body tag and add this line, which inserts code for the navigation menu:

```
<%@ include file="menu.jsp"%>
```

3. In the Navigation Pane, find the `ApplicationResources.properties` file in the `resources` directory and double-click it to open it in the Edit Pane.
4. Look for the resource text strings that begin with `site`:

```
site.title=Words of Wisdom
site.greeting=Welcome to Words of Wisdom
site.info=Return each day to read a proverb<br>from somewhere around the
world...
```

You can use these property keys in `index.jsp` to refer to these text strings instead of putting the text directly in the file. This makes it easy to translate the site if you want to.

5. Switch back to `index.jsp` in the Edit Pane.

There are several ways to switch among open files. You can use the:

- Tabs in the Edit Pane
- Documents menu
- Project tab of the Navigation Pane—double-click the file again in the `jspx` directory

6. Replace the text between the title element's start and end tags with this Struts message tag:

```
<strutsbean:message key="site.title"/>
```

7. Replace the content between the line that includes menu.jsp and `</body>` with a heading and two paragraph elements. This code contains Struts message tags that display the site.info text and error messages. The font tags are optional:

```
<h1>
<font face="Verdana, Arial, Helvetica, sans-serif" size="3">
  <strutsbean:message key="site.greeting"/>
</font>
</h1>

<p>
<font face="Verdana, Arial, Helvetica, sans-serif" size="-1">
<strutsbean:message key="site.info"/>
</font>
</p>

<p>
<strutshtml:errors />
</p>
```

8. For fun, add a color to the body tag. Replace `<body>` with:

```
<body bgcolor="#FF6633">
```

If you don't like this warm orange, specify whatever color value you like.

The resulting file should look like this:

```
<!--
  index.jsp

  Generated by SilverStream XSLT Code Generator, version 1.0.
  This generated source file may be freely modified.

-->
<%@ page language="Java"
  session="true"
  isThreadSafe="true"
  contentType="text/html; charset=ISO-8859-1" %>
<%@ taglib uri="struts-html.tld" prefix="strutshtml" %>
<%@ taglib uri="struts-bean.tld" prefix="strutsbean" %>

<html>
<head>
  <title>
    <strutsbean:message key="site.title"/>
  </title>
</head>

<body bgcolor="#FF6633">

<%@ include file="menu.jsp"%>
```

```

<h1>
<font face="Verdana, Arial, Helvetica, sans-serif" size="3">
  <strutsbean:message key="site.greeting"/>
</font>
</h1>

<p>
<font face="Verdana, Arial, Helvetica, sans-serif" size="-1">
<strutsbean:message key="site.info"/>
</font>
</p>

<p>
<strutshtml:errors />
</p>

</body>
</html>

```

9. Save the file.
10. Close the file. You can select **File>Close** or click the **Close** button in the Edit Pane.

Building and archiving

On the Project menu, Workbench has commands for building (compiling) individual files and for rebuilding all the files in the project before you generate the archive.



EXERCISE 4-4: Compile the Java code and generate the archive file

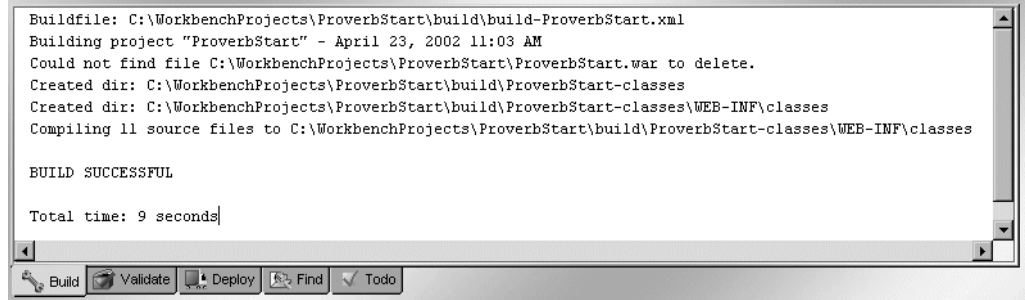
In this exercise you will compile all the code in the project. If there are no errors, you can generate the archive.

1. Select **Project>Rebuild All** from the menu.

NOTE If a file open in the Edit Pane has been changed, Workbench saves it. You can change this behavior in Workbench Preferences.

On the Build tab of the Output Pane at the bottom of the Workbench window, you see messages reporting progress as well as any warnings and errors that occur.

If your classpath includes all the required JARs, there should be no errors. If there are errors, you can double-click the error line in the Output Pane to go to the file and line that caused the problem.



```
Buildfile: C:\WorkbenchProjects\ProverbStart\build\build-ProverbStart.xml
Building project "ProverbStart" - April 23, 2002 11:03 AM
Could not find file C:\WorkbenchProjects\ProverbStart\ProverbStart.war to delete.
Created dir: C:\WorkbenchProjects\ProverbStart\build\ProverbStart-classes
Created dir: C:\WorkbenchProjects\ProverbStart\build\ProverbStart-classes\WEB-INF\classes
Compiling 11 source files to C:\WorkbenchProjects\ProverbStart\build\ProverbStart-classes\WEB-INF\classes

BUILD SUCCESSFUL

Total time: 9 seconds
```

The screenshot shows a build console window with a toolbar at the bottom containing buttons for Build, Validate, Deploy, Find, and Todo. The console text indicates a successful build of the 'ProverbStart' project, including directory creation and compilation of 11 source files.

2. Select **Project>Build and Archive** to generate the archive file.

About compiling There are several Build menu items on the Project menu. You might want to compile a file open in the Edit Pane, build only files that have changed, or rebuild all files. To learn more about the building and archiving commands, see the chapter on projects and archives in the *Tools Guide*.

Working with 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 all the elements it can include, 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 built the archive in EXERCISE 4-4: "Compile the Java code and generate the archive file". If it isn't already built, Workbench can build it for you.

For the Proverbs application, you need to add parameters for the Struts controller servlet, servlet mappings for Struts URLs, a default welcome page, and the Struts tag libraries.



EXERCISE 4-5: Begin editing the deployment descriptor

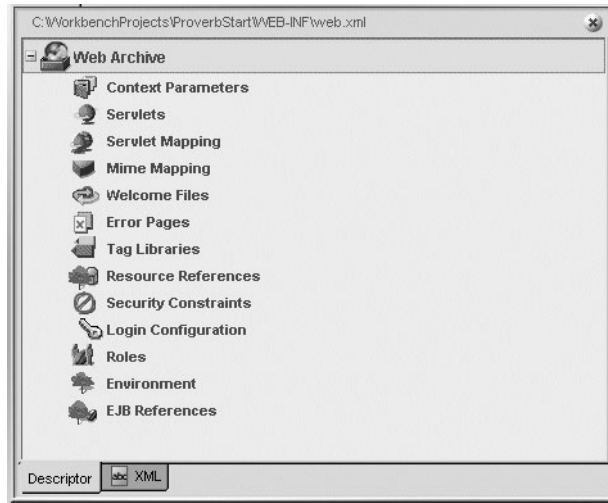
In this exercise you will open web.xml and add a reference to the Struts action controller servlet.

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

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

2. If Workbench displays the **Select Build Option** dialog, accept the defaults and click **OK**. Workbench opens web.xml in the Edit Pane.
3. Click the **Descriptor** tab.

The editor shows the types of information the descriptor can include.



4. Click the **XML** tab to see the raw content of the descriptor. So far, there isn't much.
5. Switch back to the **Descriptor** tab.
6. Right-click **Web Archive** and select **Properties**.
7. Specify **5** (minutes) as the session timeout.
8. Right-click **Servlets** and select **Add** from the popup menu.
A new **Untitled** entry appears indented under **Servlets**.

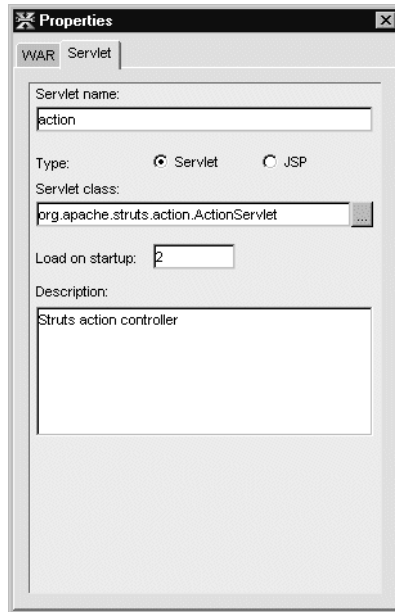


9. Right-click **Untitled** and select **Properties**.

10. In the Servlet property sheet, make these settings:

Option	Value
Servlet name	action
Type	Servlet
Servlet class	org.apache.struts.action.ActionServlet
Load on startup	2
Description	Struts action controller

The property sheet looks like this:



11. Save the file.

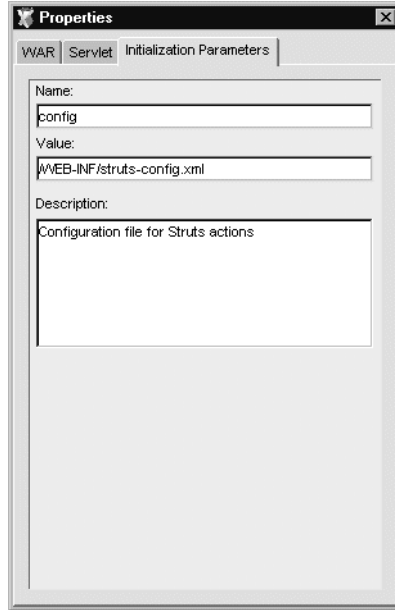


EXERCISE 4-6: Add initialization parameters for the servlet

In this exercise you will add several parameters for the Struts controller servlet. The parameters you need depend on the requirements of the individual servlet. `web.xml` is still open in the Edit Pane.

1. In the Servlets section under the newly added action servlet, right-click **Initialization Parameters** and select **Add** from the popup menu.
If the elements under **action** aren't visible, click the plus sign beside **action** to expand the list.
2. Click the newly added **Untitled** to highlight it. (If the Property Inspector isn't open, right-click **Untitled** and select **Properties**.)
Its properties display in the Initialization Parameters property sheet.
3. In the property sheet, specify these settings:

Option	Value
Name	config
Value	/WEB-INF/struts-config.xml
Description	Configuration file for Struts actions



4. Add four more parameters by right-clicking **Initialization Parameters**, selecting **Add**, then highlighting **Untitled** to set the properties.

For the first parameter, specify these values in the property sheet:

Option	Value
Name	mapping
Value	com.proverb.ProverbActionMapping

For the second, specify these values:

Option	Value
Name	application
Value	com.proverb.ApplicationResources

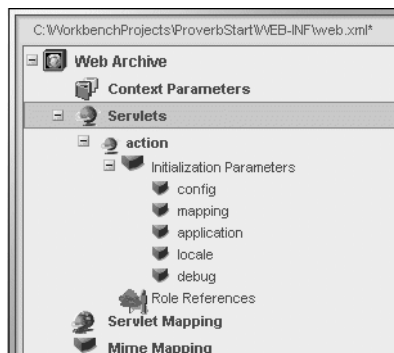
For the next, specify these values:

Option	Value
Name	locale
Value	true

For the last, specify these values:

Option	Value
Name	debug
Value	3

The servlet section of the editor looks like this when you've added all the parameters:



5. Save the file.



EXERCISE 4-7: Add a servlet mapping

In this exercise you will add a servlet mapping that tells the Struts action servlet how to handle specific URLs.

1. With web.xml open in the Edit Pane, right-click **Servlet Mapping** and select **Add** from the popup menu.

2. Select the new **Untitled** mapping and make these settings in the property sheet (if the Property Inspector isn't open, you know how to open it):

Option	Value
Servlet name	action
URL pattern	*.do



EXERCISE 4-8: Specify the project's default page

In this exercise you will specify that `index.jsp` is the page to be displayed if the user doesn't specify a specific page.

1. With `web.xml` open in the Edit Pane, right-click **Welcome Files** and select **Add** from the popup menu.
2. Select the new **Untitled** item and in the property sheet specify **index.jsp** as the Welcome File.
3. Save the file.



EXERCISE 4-9: Add tag libraries

In this exercise you will identify the tag libraries that the Struts application uses and where they are in the archive.

1. With `web.xml` still open in the Edit Pane, add three new items under **Tag Libraries**. The technique is the same as for the initialization parameters: for each one, right-click **Tag Libraries** and select **Add** from the popup menu. Then select the new **Untitled** item and set the values in the Tag Library property sheet. For the first tag library, specify these values:

Option	Value
Tag library URI	struts-html.tld
Tag library location	/WEB-INF/struts-html.tld

For the next, specify these values:

Option	Value
Tag library URI	struts-logic.tld
Tag library location	/WEB-INF/struts-logic.tld

For the last, specify these values:

Option	Value
Tag library URI	struts-bean.tld
Tag library location	/WEB-INF/struts-bean.tld

2. Save and close web.xml.



EXERCISE 4-10: Rebuild the archive

In this exercise you will rebuild the archive so that it includes the newly edited version of web.xml.

1. Select **Project>Build and Archive** to regenerate the archive file.
2. Check the Output Pane for messages.

Deploying the project

Using Workbench, you can create J2EE applications for any J2EE application server. For some application servers, Workbench supports deployment directly with a Deploy Archive menu item. For other application servers, you can build your archive in Workbench and use your server's deployment tools to deploy. As long as you stick to J2EE standards and avoid server-specific code, archives built by Workbench are completely J2EE-compatible and can be deployed to any J2EE server.

To deploy you'll need to do these tasks:


1. Define a server profile
2. Prepare server deployment information expected by your server, if any
3. Specify deployment settings

4. Deploy the archive

If you've done another Workbench tutorial most of your deployment setup has already been done. The next exercise gives you the main steps and provides project-specific information for deploying this project. To read detailed deployment instructions for directly supported servers, see Workbench Deployment Instructions.




EXERCISE 4-11: Deploy the project

1. 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.
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.

Server	Where	Option and value or file contents
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 — ProverbStart</p> <p>Server Profile — Select the profile you defined in the previous step from the dropdown list box</p> <p>Session timeout — 5 minutes (set in the deployment descriptor; not overridden here)</p> <p>URLs — ProverbStart</p> <p>You can specify one or more relative URLs for the Web application; Workbench automatically provides the archive name as the first URL</p> <p>Excluded JSPs — menu.jsp</p> <p>Menu.jsp is an incomplete fragment included in the other JSP pages. It doesn't have the appropriate headers to compile correctly. You can either ignore the errors it causes (a Deployment Settings option) or use the Excluded JSPs property to prevent the server from trying to compile it.</p> <p>Uses JARs — <i>Leave blank</i></p>

Server	Where	Option and value or file contents
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>ProverbStart</display-name> <context- root>ProverbStart</context-root> </web> </j2ee-ri-specific-information></pre>
Jakarta Tomcat	—	Tomcat is a servlet container and does not support the database access required of the Proverbs application
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/wl s610/dtd/ weblogic-web-jar.dtd"> <weblogic-web-app> <description> Proverbs Web application </description> <weblogic-version> </weblogic-version> </weblogic-web-app></pre>
IBM WebSphere	—	—
Oracle9iAS	—	—

 For more information and exercises with detailed steps, select the section for your server in Workbench Deployment Instructions.

- Specify deployment settings for your server by selecting **Project>Deployment Settings**. 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	—
Tomcat	—
BEA WebLogic	WebLogic Application Name — ProverbStart; 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 — ProverbStart; 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 detailed instructions, 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 in 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.

Testing the application

A WAR is a Web application, so you test it in your Web browser.



EXERCISE 4-12: Test the application in the browser

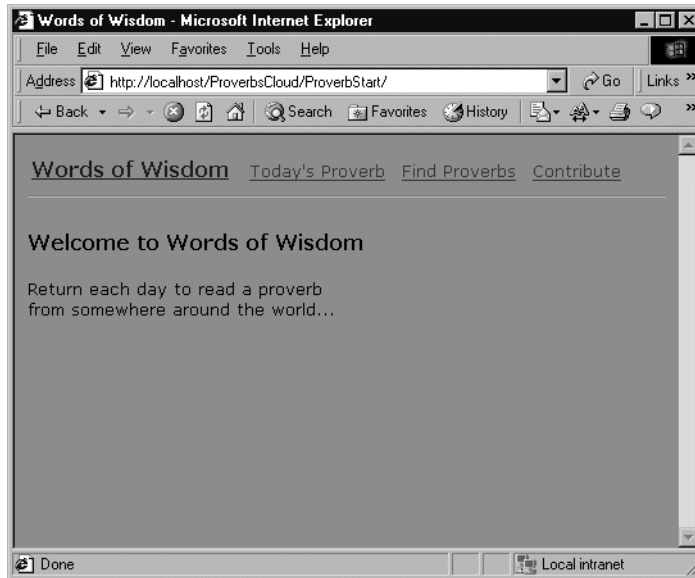
1. If you're already reading this tutorial in the browser, open a new browser window. If not, start the browser of your choice.
2. Go to the URL for your Web application. It will generally include these parts:

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/ProverbsCloud/ http://www.mydomain.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. For some servers, it is the name of the J2EE archive—in this case, the same value	ProverbStart/
Web page	(Optional) Name of the JSP page you want to view; the Welcome page displays if you omit this part of the URL	index.jsp

For example, if the application is deployed to a local SilverStream server in a database called ProverbsCloud and the URL in the deployment plan is ProverbStart, the case-sensitive URL would be:

```
http://localhost/ProverbsCloud/ProverbStart/index.jsp
```

You see the index.jsp page with the text from the ApplicationResources.properties file. The links in the navigation menu don't work yet; you'll add those pages in later lessons.



Summary of what you've done

Developing the application In this lesson you built these parts of the Proverbs tutorial application:

- Initial page of the application, implemented as a JSP page
- Resource file for displayed text
- Deployment descriptor
- Server profile for deployment
- Server-specific deployment information; for SilverStream servers, a deployment plan

Using Workbench tools You used these tools in Workbench:

- JSP Wizard (File>New)
- Deployment Descriptor Editor
- Menu items on the Project menu for building and archiving
- Server profiles (Edit>Profiles)
- Deployment Plan Editor (File>New or right-click on project and select Open Deployment Plan), XML Editor, or Text Editor
- Deployment (Project>Deploy Archive)

Next lesson At this point you've completed the basics for developing a Web application (WAR) using Workbench. Feel free to stop here and try out what you've learned. You can try creating your own project or experiment with changing parts of the Proverb project.

The lessons in Part II, "Forms and Data in a Web Application" show how to use the database of proverb data with your application.

Part II Forms and Data in a Web Application

This part teaches you how to write code in a Struts Web application that accesses data via the J2EE application server's connection pool and how to define forms that display that data.

The lessons are:

- Lesson 5, "Setting Up Database Access"
- Lesson 6, "Defining an Action That Displays Data"
- Lesson 7, "Defining a Form and Results Page"
- Lesson 8, "Defining a Form for Database Update"

In this part you will learn more about Struts and how it makes handling forms easier in the HTTP environment. Lesson 5, "Setting Up Database Access" sets up the data access classes and configures the Web application to access the server's connection pool.

After the data access is set up, you can choose to do one or all of the remaining lessons. Each of these lessons implements one of the items on the Proverbs navigation menu using Struts techniques. Most of the Java code is already written, and you will use Workbench to look at that code and to edit XML configuration files.

5 Setting Up Database Access

What you will learn

This lesson illustrates the code for accessing the data source and shows how to add database connection information to the application's configuration files.

You will learn about:

- Making the data source available to the application

What you will do

1. Add a resource reference to the deployment descriptor
2. Identify the database in the server deployment information
3. Extend the Struts ActionServlet to get the data source during initialization
4. Change the class for the application's startup servlet

How long will it take? About 10 minutes

NOTE If you use the SilverStream eXtend Application Server, it must be running when you edit the deployment plan. If you use another application server, its tools for specifying server deployment information may require the server to be running.

Making the data source available to the application

To make the database available to the application, you need to:

- In the deployment descriptor, identify a resource name for the database
- In the server's deployment information, associate the resource name with an actual database
- In the application code, get a DataSource object from the server's connection pool when the application servlet is initialized and save it in the servlet context

You will do this in the next exercises.

Resource references in the deployment descriptor

In the deployment descriptor you can identify names that your application uses to connect with external services.

Resource reference The name for a database connection is in the Resources section of the deployment descriptor. In the Proverbs application, you'll call it:

```
jdbc/ProverbsDB
```

When you deploy, this JNDI name is stored in the application server's naming service.

NOTE JNDI is the Java Naming and Directory Interface. Application servers include a naming service for finding objects in a distributed computing environment.

Environment variable The application doesn't refer to the resource reference directly; instead, an environment variable stores the name of the resource reference:

```
jndi-datasource-name
```

This indirection avoids hardcoding the application-specific database identifier in the Java code. The code only needs to know the environment variable.

Getting the environment variable Constants in the Java code identify the naming service context and the environment variable:

```
// Context in naming service for environment entries
final static String JAVA_COMPONENT_ENV = "java:comp/env";
// Environment entry for data source name
final static String JNDI_DATASOURCE_NAME = "jndi-datasource-name";
```

The code concatenates the strings to look up the environment variable; it uses the variable to get the resource name; then it looks up the resource in the naming service.



EXERCISE 5-1: Add a resource reference to the deployment descriptor

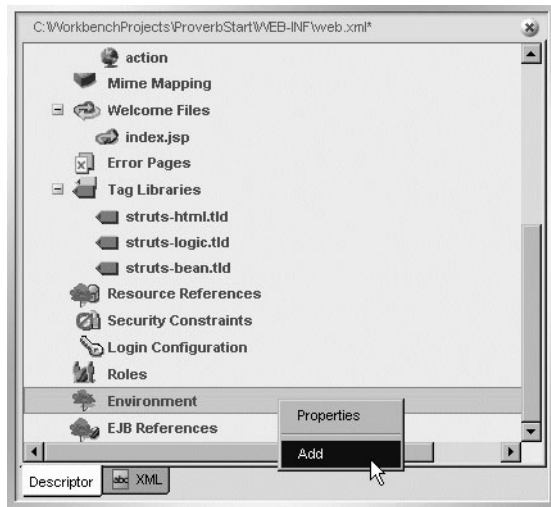
In this exercise you will define a resource reference for the database and an environment entry that identifies the name of that resource.

1. Start Workbench if it isn't already running, then open the **ProverbStart** project.

TIP If you have worked on the tutorial recently, open ProverbStart by selecting **File>Recent Files** and choosing it from the list.

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

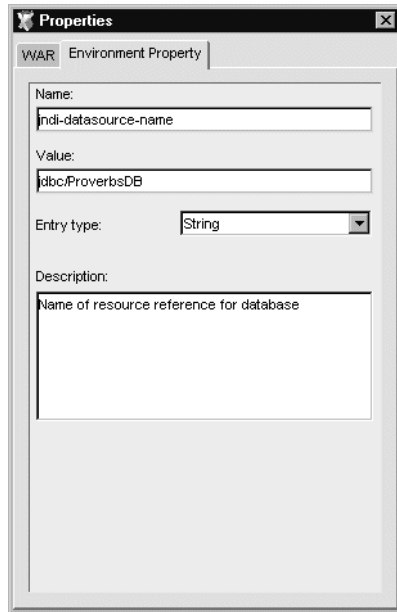
3. If Workbench displays the **Select Build Option** dialog, accept the defaults and click **OK**. Workbench builds the project, if requested, then opens the descriptor.
4. In the Edit Pane, select the **Descriptor** tab if it isn't selected already.
5. Right-click **Environment** and select **Add**.



6. Right-click **UntitledEnvironmentProperty** and select **Properties**.
7. In the Environment Property property sheet, specify this information:

Option	Value
Name	jndi-datasource-name
Value	jdbc/ProverbsDB
Entry type	String
Description	Name of resource reference for database

The property sheet looks like this:



8. In the Edit Pane, right-click **Resource References** and select **Add**.



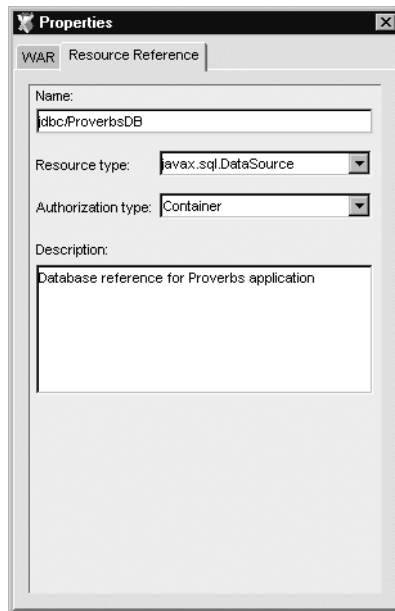
9. Select **UntitledResourceReference**.

If the Property Inspector is still open, it displays properties for the new resource reference. If you closed the Property Inspector, open it by right-clicking the new entry and selecting **Properties**.

10. In the Resource Reference property sheet, specify this information:

Option	Value
Name	jdbc/ProverbsDB
Resource type	javax.sql.DataSource
Authorization type	Container
Description	Database reference for Proverbs application

The property sheet looks like this:



11. Close the Property Inspector.
12. Save the descriptor file and close it.

Identifying the database in the server deployment information


In the server deployment information, you associate a database from the server's connection pool with the resource reference in the deployment descriptor. The deployment information connects the virtual resource with a real data source.

The way the server handles deployment information and resource references for databases depends on the application server. Workbench provides an editor for creating a Workbench deployment plan for the SilverStream eXtend Application Server. For other servers, use your server tools and documentation to find out how to associate the descriptor's resource reference with the database of proverbs (which you set up in Lesson 2, "Setting Up Your Data Source").



EXERCISE 5-2: Identify the database in the server deployment information

In this exercise you'll add XML elements or other formatted data to the file that holds your server's deployment information.

- Use the following information to edit 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	Where	Option and value or file contents
SilverStream	<p>Edit the SilverStream deployment plan. When opening the deployment plan, make sure you have Workbench build the project so it picks up the changes you just made to the deployment descriptor. Also, make sure the SilverStream server is running in order to get a list of connection pools.</p> <p>Open the property sheet for the Resource Reference called jdbc/ProverbsDB</p>	<p>Connection pool — Specify the database of proverbs that you've already added to the server. If you are using the provided Cloudscape database, specify ProverbsCloud. (If you defined your project as a J2EE 1.3 project and are deploying to a SilverStream 4.x server, specify /JDBC/ProverbsCloud.)</p>
Sun Reference Implementation	<p>Edit sun-j2ee-ri.xml, which is included in the archive in the META-INF directory.</p> <p>Add this XML inside the web element after the context-root element</p>	<pre><resource-ref> <res-ref-name> jdbc/ProverbsDB </res-ref-name> <jndi-name> ProverbsCloud </jndi-name> </resource-ref></pre> <p>For ProverbsCloud, substitute the JNDI name you used when you installed the database of proverbs on the server.</p>

Server	Where	Option and value or file contents
Jakarta Tomcat	—	Tomcat is a servlet container and does not support the database access required of the Proverbs application
BEA WebLogic	Edit weblogic.xml, which is included in the archive in the WEB-INF directory. Add this XML after the weblogic-version element.	<pre><reference-descriptor> <resource-description> <res-ref-name> jdbc/ProverbsDB </res-ref-name> <jndi-name> ProverbsCloud </jndi-name> </resource-description> </reference-descriptor></pre> <p>For ProverbsCloud, substitute the JNDI name you used when you installed the database of proverbs on the server.</p>
IBM WebSphere	—	—
Oracle 9iAS	—	—

Getting the data source when the application starts

After the resource for the data source is set up, you need to write code that uses the server's naming service and connection pool to get a `DataSource` object. In the Proverbs application this code is in the `ProverbActionServlet`, which extends the standard `ActionServlet` provided by Struts. It overrides the servlet's `init()` method to get the `DataSource`.

You also need to tell the server to use this new servlet class. You do that in the deployment descriptor.



EXERCISE 5-3: Extend the Struts ActionServlet to get the data source during initialization

In this exercise you will look at the `ProverbActionServlet` class, which extends `ActionServlet`.

1. In the Archive layout view of the Workbench Navigation Pane, expand the **WEB-INF** directory down to the `proverb` directory and highlight **proverb**.
2. Double-click **ProverbActionServlet.java** to open it in the Edit Pane.

3. Notice that the class declaration extends **ActionServlet**.
4. Notice that the **init()** method calls the parent's **init()** method to perform the standard initialization tasks.

```
super.init();
```

5. Look at the code that accesses the server's naming service to get a **DataSource** object. It stores the **DataSource** object in the servlet context.

Here's the code (with comments removed):

```
Context ic = new InitialContext();
Context env = (Context) ic.lookup(Constants.JAVA_COMPONENT_ENV);
String dsName = (String) env.lookup(Constants.JNDI_DATASOURCE_NAME);
DataSource ds = (DataSource) env.lookup(dsName);
getServletContext().setAttribute(Constants.DB_KEY, ds);
```

6. (Optional) Open **Constants.java** to see the values of the constants used in this code.
7. Close the files.

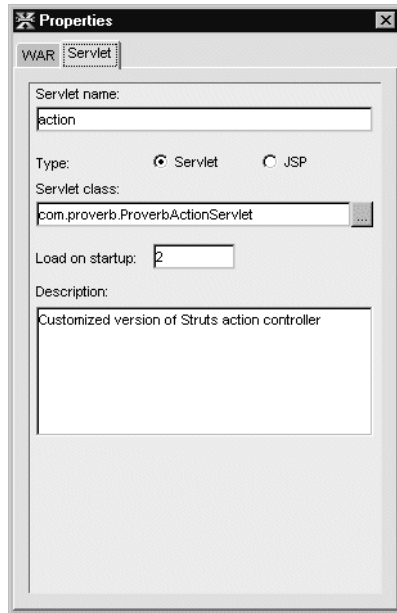


EXERCISE 5-4: Change the class for the application's startup servlet

In this exercise you will edit the deployment descriptor to change the application's startup servlet.

1. In the Navigation Pane, right-click the project file **ProverbStart.spf** and select **Open Deployment Descriptor** from the popup menu.
2. If Workbench displays the **Select Build Option** dialog, accept the defaults and click **OK**. Workbench builds the project, if requested, then opens the descriptor.
3. In the Servlets section, right-click **action** and select **Properties**.
4. In the Servlet property sheet, change the servlet class to **com.proverb.ProverbActionServlet**. You can click the ellipses button and select the class from a list.
5. Change the description to **Customized version of Struts action controller**.

The property sheet looks like this:



6. Close the Property Inspector.
7. Save and close the descriptor.

Summary of what you've done

Developing the application In this lesson you built or learned about these parts of the Proverbs tutorial application:

- Resource reference in the deployment descriptor for naming an identifier for the data source
- Resource reference in the server deployment information that identifies the data source
- Customized controller servlet

Using Workbench tools You used these tools in Workbench:

- Deployment Descriptor Editor
- Deployment Plan Editor, XML Editor, or Text Editor

Next lesson In the next lesson you will learn how to code and configure a Struts action.

6

Defining an Action That Displays Data

What you will learn

In this lesson you will learn about Struts custom tags for displaying data and Java code for handling a Struts action. You will also specify what the controller needs to know about the action in the configuration file.

You will learn about:

- Querying the database
- Struts support for an action
- Telling the controller about a Struts action
- Retrieving data in the Action class
- Struts tags for displaying data
- Deploying and testing data access

What you will do

1. Retrieve data from the database
2. Define the action in the Struts configuration file
3. Use an Action class to set up the data for a JSP page
4. Display the retrieved data in a JSP page
5. Deploy the application
6. Test today.jsp

How long will it take? About 10 minutes

NOTE In this lesson you will need to run your J2EE application server for the deployment and testing exercises.

Querying the database

One of the activities of the Proverbs application is to display today's proverb. When the user chooses the Today's Proverb link, the URL **today.do** is sent to the server. The controller recognizes this URL and invokes Struts processing. The controller calls a method in the TodayAction class that sets up a data object with the appropriate proverb. Based on the return value, the controller selects a JSP page to display this data to the user.

To handle the proverb data, the application has:

- A **Proverb bean class** that holds the data for one proverb. Its properties correspond to the columns in the proverbs database table.
- A **ProverbDataAccess class** whose methods query the database and return a Collection of one or more Proverb beans
- A **TodayAction class** that makes the data available by calling the `getTodaysProverb()` method in ProverbDataAccess
- A **JSP page** that uses the Collection object to display the Proverb bean's property values
- An **action element** in the Struts configuration file tying the TodayAction class and `today.jsp` to a request URL

To keep the business logic separate from the application flow, the data access methods are isolated in the ProverbDataAccess class; you'll see that none of the application's Action classes query the database directly.



EXERCISE 6-1: Retrieve data from the database

In this exercise you will look at how the ProverbDataAccess class queries the database and uses the Proverb class to build a Java Collection of the retrieved data.

1. In the Workbench Navigation Pane, find the file **Proverb.java** and open it.
In Archive layout it is in the proverb folder under the **WEB-INF/classes** directory. In Source layout it is under the **src** directory.
2. In the Edit Pane, notice these pieces of code, which make the Proverb object a bean:
 - The class implements the **java.io.Serializable** interface
 - The properties are defined as three **instance variables**; they correspond to columns in the proverbs database table
 - Each instance variable has a **get** and **set** method
3. Open the file **ProverbDataAccess.java** and find the **getTodaysProverb()** method.

4. Notice how the code:
 - Uses constants to build the **SQL query string**; the column names don't need to be specified here
 - Instantiates a **Proverb** object and sets its properties from the retrieved data
 - Adds the Proverb object to the proverbs **ArrayList** and returns the ArrayList as a **Collection**
5. Close the files.

Struts support for an action

Review Lesson 1, “Architecture of an MVC Application” described how Struts uses a controller to handle URLs. The configuration file **struts-config.xml** configures the controller and tells it what processing to use for each action. Java classes do the processing, and JSP pages display the results to the user.

You configure and support an action in a Struts application in these places:

Place	Description
web.xml (deployment descriptor)	The servlet mapping specifies the URLs the controller handles. In Proverbs, the controller handles all URLs in the form *.do (in EXERCISE 4-7: “Add a servlet mapping” you already specified the servlet mapping for the URL)
struts-config.xml	An action element identifies the action class that responds to the URL and the JSP page to display
Action class	A Java class extending the Struts Action class performs setup for the JSP page
ActionForm class	If the JSP page contains forms, a Java class extending the Struts ActionForm class is instantiated to hold the data for each form field

Place	Description
ActionMappings class	A Java class extending the Struts ActionMappings class defines properties for the forwarding keywords used by all the actions
JSP pages	The data to display to the user may depend on what happens in the Action class's code; the Action class returns a value to the controller indicating which JSP page to display

In Lesson 3, “Working with Projects and Archives” you added the Struts JAR and tag libraries to the project and assigned them to a location in the archive. They are required for handling the Struts code.

In this lesson You'll configure the today action and look at the associated Action class and JSP page. There are no form classes for this action. In this lesson you will:

- Add the today action to struts-config.xml
- Look at the TodayAction class that extends Action and see how it accesses data
- Look at today.jsp to see how the Struts tags access the data set up by TodayAction

Telling the controller about a Struts action

The Struts configuration file contains action elements, which specify what happens when the controller receives a URL with a particular action path. The action element specifies what processing occurs and what page to display next. The controller reads this file to find out what it is supposed to do.

An action element looks like this:

```
<action path="/actionname"
        type="actionclass" >
    <forward name="keyword" path="/target.jsp"/>
    <forward name="keyword" path="/target.jsp"/>
</action>
```

The table describes how the attributes and elements for an action are used in the today action:

Attribute or element	Purpose	Values in the today action
path	Keyword in the URL that invokes this processing. The path always begins with /. The servlet mapping in web.xml specifies how the path keyword is combined to form a URL.	The complete path is actionname.do , which is set up in web.xml. The path /today tells the controller what should happen when it receives the URL today.do .
type	Action class that sets up the application environment for displaying a Web page	The type com.proverb.TodayAction tells the controller to call the perform() method of the TodayAction class.
forward	JSP pages to be displayed as a result of this action. The Action class chooses which forward name to return to the controller. The name and path attributes associate a forward name with a JSP page.	The Proverbs application uses forward values of success and failure . When perform() returns a forward value, the controller displays the JSP page specified by the named forward element's path attribute.

A table laying out the attribute values for each action is a good planning tool for creating the specifications for an application, as shown in “How Struts handles actions” on page 16. You can fill out a table in that format for the today action if you want.



EXERCISE 6-2: Define the action in the Struts configuration file

In this exercise you will add an action to the configuration file that tells the controller what to do when it gets the URL **today.do**.

NOTE You can copy the XML for this exercise from the file **CutAndPasteCode.txt** in the **Workbench-install-dir/docs/tutorial/TutorialFiles/proverbs** directory. Select File>Open to open the file.

1. In the Workbench Navigation Pane, find and open **struts-config.xml**. It's in the **WEB-INF** directory.

2. Inside the action-mappings element, enter this XML:

```
<action path="/today"
        type="com.proverb.TodayAction" >
    <forward name="success" path="/today.jsp"/>
    <forward name="failure" path="/index.jsp"/>
</action>
```

3. Save and close **struts-config.xml**.
4. (Optional) Open **web.xml** and look at the servlet mapping. It specifies that the action servlet handles any URL in the form ***.do**. Then close the file.

Retrieving data in the Action class

The processing for TodayAction is simple: it gets the application's DataSource object, which was stored in the servlet context when the controller servlet was initialized; then it calls a method in ProverbDataAccess to get the proverb of the day.



EXERCISE 6-3: Use an Action class to set up the data for a JSP page

In this exercise you will look at the TodayAction class, which gets the Collection from ProverbDataAccess and makes it available to the JSP page.

1. In the Workbench Navigation Pane, find the file **TodayAction.java** and open it.
NOTE Do you remember where to find the Java files? In Archive layout they're under the WEB-INF/classes directory. In Source layout they're under the src directory.
2. Find the code in the **perform()** method that gets the DataSource object from the servlet context:

```
ServletContext ctxt = servlet.getServletContext();
DataSource ds = (DataSource) ctxt.getAttribute(Constants.DB_KEY);
```

3. Look at the code that instantiates the **ProverbDataAccess** class and calls the **getTodaysProverb()** method. The constructor expects the DataSource object as an argument.

```
ProverbDataAccess pda = new ProverbDataAccess(ds);
if (pda != null)
    pvbs = pda.getTodaysProverb();
```

4. Find the code that saves the pvbs Collection in the request:

```
request.setAttribute(Constants.PVB_KEY, pvbs);
```
5. Open **Constants.java** to find out the value of **PVB_KEY**. The JSP page uses that value to refer to the collection.

6. Notice that when the data retrieval is successful, the `perform()` method returns an `ActionForward` value of **success**:

```
fwd = "success";
...
return (mapping.findForward(fwd));
```

This tells the controller which JSP page to display.

7. Close the files.

Struts tags for displaying data

Struts provides custom tags that can apply HTML formatting to a collection of data. You don't have to know ahead of time how many items you'll get.

In `today.jsp` you'll find these tags:

Tag	Description
iterate	Iterate repeats the enclosed statements for each object in the specified collection. The name attribute specifies the collection that <code>TodayAction</code> puts in the request context. Tags inside <code>iterate</code> refer to the collection using the value of id . The JSP page <code>today.jsp</code> uses <code>iterate</code> to identify the collection being processed; it doesn't matter that there is only one item in the collection.
write	The <code>property</code> attribute for the <code>write</code> tag specifies a property of the <code>Proverb</code> object whose value is written as text in the JSP page.
present	Struts checks whether the property you specify has a value. If so, Struts processes the enclosed tags and text. In <code>today.jsp</code> , present checks whether the translation property has a value; and if not, it omits the text label too.



EXERCISE 6-4: Display the retrieved data in a JSP page

In this exercise you will see how `today.jsp` accesses the collection and displays the daily proverb.

1. Double-click **today.jsp** to open it.

2. In the Edit Pane, find the **strutslogic:iterate** tag:

```
<strutslogic:iterate name="ProverbCollection" id="pvb">
```

It uses the ProverbCollection attribute that TodayAction set in the request.

Do you recognize the value of the name attribute? It's the value of PVB_KEY in Constants.java.

3. Inside the iterate tag, find a **write** tag. There are several.

```
<strutsbean:write name="pvb" property="text" />
```

In each write tag, the property attribute refers to a property in the Proverb class.


4. Close the file.

Converting plain text to message tags The h2 text and the text labels for the data are just English text in this file. If you wanted to internationalize this application, do you know how you would fix this with Struts message tags?

The text for the heading is already part of the ApplicationResources.properties file. You can replace the text in the heading with this message tag:

```
<strutsbean:message key="page.today.title"/>
```

For the data labels, you need to add keys and their text to the resources file, then refer to those keys in the JSP page.

 For more information about using the Struts message tag, see “How Struts enables internationalization and localization” on page 13.

Deploying and testing data access

You've reached the point where you can test the data access and see today.jsp display the proverb data. You did the deployment setup the first time you deployed when you defined a server profile and specified deployment settings in Lesson 4, “Deploying and Testing the Welcome Page”. Now all you need to do is build and deploy.



EXERCISE 6-5: Deploy the application

1. In Workbench, select **Project>Build and Archive**.
There should be no errors in the Build tab of the Output Pane.
2. Select **Project>Deploy Archive** from the menu.

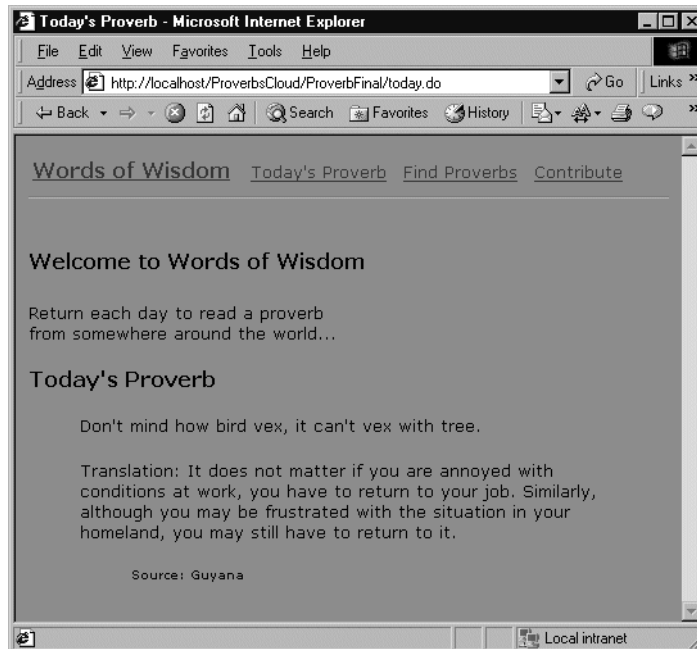
The Output Pane displays brief build output, then switches to the Deploy tab and shows progress messages. There should be no errors.



EXERCISE 6-6: Test today.jsp

1. If your browser is running, open a new browser window. If not, start your browser now.
2. Enter the URL for the application. A typical URL for a SilverStream server might be:
`http://localhost/ProverbsCloud/ProverbStart/index.jsp`
3. Click **Today's Proverb** in the application menu.

If the page displays a proverb, congratulations. You've successfully set up the today action.



If it doesn't work If you see no proverb but errors instead, you'll need to reexamine the server deployment information, deployment descriptor, and Struts configuration file to make sure they are correct. You can also check the code that:

- Accesses the server's naming service in `ProverbActionServlet`
- Does the database query in `ProverbDataAccess`

- Defines constants for the SQL statement in Constants.java

TIP Check the server console for error messages. If there are server-specific errors, the code may need editing.

Summary of what you've done

Developing the application In this lesson you built or learned about these parts of the Proverbs tutorial application:

- ProverbDataAccess class with methods that access the database
- Action element for the today action in struts-config.xml
- TodayAction class for handling the today.do URL
- JSP page called today.jsp with custom tags for displaying proverb data

Using Workbench tools You used these tools in Workbench:

- Edit Pane
- Building and archiving (Project>Build and Archive)
- Deployment (Project>Deploy Archive)

Next lesson In the next lesson you will learn how to get search criteria from the user in a form and how to use Struts support for form processing.

7 Defining a Form and Results Page

What you will learn

In this lesson you will learn about Struts custom tags for displaying data in JSP pages and about Java code for handling a form. You will also learn what the Struts controller needs to know about forms in the configuration file.

You will learn about:

- Two actions for one form
- Setting up the form
- Processing for the actions
- Displaying the retrieved data
- Deploying and testing the form

What you will do

1. Define two actions in the Struts configuration file
2. Examine the form elements in the JSP page
3. Examine the SelectForm class
4. Examine the SelectAction class
5. Examine the JSP pages that show the results of the search
6. Deploy the application
7. Test the Find Proverbs activity

How long will it take? About 10 minutes

NOTE In this lesson you will need to run your J2EE application server for the deployment exercise.

Two actions for one form

In `struts-config.xml` there are two action elements for finding proverbs: the **select** action displays the form and the **results** action processes the submitted form.

The `SelectAction` class provides processing for both actions. When the controller calls its `perform()` method, it passes an `ActionMapping` object that identifies which action is being performed.

In addition to the `Action` class, you also need an `ActionForm` class for the new form. Its name and class are defined in the `form-beans` section of the configuration file.

In Lesson 6, “Defining an Action That Displays Data” you learned about some basic attributes for actions. To review them, see “Telling the controller about a Struts action” on page 90.

Because the actions for selecting a proverb use a form, the action elements in the next exercise include some new attributes:

- **name**: the name of the form object, assigned in the form-bean element
- **scope**: how long the form object is kept; typical values are **request** and **session**
- **validate**: whether to call the form’s `validate()` method when the form is submitted
- **input**: for an action that processes a submitted form, the JSP page that displays the form



EXERCISE 7-1: Define two actions in the Struts configuration file

In this exercise you will add two actions to the configuration file.

NOTE You can copy the XML for this exercise from the file `CutAndPasteCode.txt` in the `Workbench-install-dir/docs/tutorial/TutorialFiles/proverbs` directory.

1. In the Workbench Navigation Pane, find and open `struts-config.xml`.
Do you remember where the configuration files are in your project?
2. Before the closing tag `</action-mappings>` and after the `</action>` closing tag of the today action, enter this XML for the action that displays the form:

```
<action    path="/select"
          type="com.proverb.SelectAction"
          name="selectForm"
          scope="session"
          validate="false">
  <forward name="success"    path="/select.jsp"/>
  <forward name="failure"   path="/select.jsp"/>
  <forward name="cancel"    path="/index.jsp"/>
</action>
```

3. Still inside the action-mappings element and after `</action>` for the select action, enter this XML for the action that displays the retrieved data:

```
<action      path="/results"
            type="com.proverb.SelectAction"
            name="selectForm"
            scope="session"
            input="select.jsp">
  <forward name="success"      path="/selectResults.jsp"/>
  <forward name="failure"     path="/selectFailed.jsp"/>
  <forward name="cancel"     path="/index.jsp"/>
</action>
```

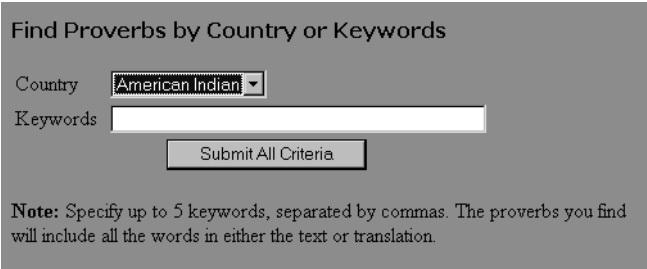
4. In the **form-beans** section at the beginning of the file, look at the **form-bean** element that defines selectForm and identifies its Java class.

```
<form-bean  name="selectForm"
            type="com.proverb.SelectForm"/>
```

5. Save and close the file.

Setting up the form

The Find Proverbs activity displays a form so users can select proverbs.



Find Proverbs by Country or Keywords

Country

Keywords

Note: Specify up to 5 keywords, separated by commas. The proverbs you find will include all the words in either the text or translation.

Using Struts tags to define a form

Struts provides custom tags that provide functionality parallel to the HTML form tags. In addition to displaying the various input fields and buttons, the tags have attributes that associate those fields with Struts form processing.

In **select.jsp**, the custom form tags are laid out using standard HTML table tags.



EXERCISE 7-2: Examine the form elements in the JSP page

In this exercise you will look at how the form implements properties and initializes data.

1. In the Workbench Navigation Pane, find `select.jsp` and open it in the Edit Pane.
2. Find the **form** custom tag. Its **action** attribute specifies the URL that is submitted to the controller, and its **name** attribute identifies the form class that will store the submitted data (described below).

```
<strutshtml:form action="results.do" name="selectForm"
type="com.proverb.SelectForm">
```

3. Find the tags for the **keywords** field. You'll see a text field and its label. The label text comes from the resources file. Here's the code without the table tags:

```
<strutsbean:message key="page.select.keywordslabel"/>
<strutshtml:text property="keywords" size="40" maxlength="40"/>
```

4. Find the tags for the **country** field.

```
<strutsbean:message key="page.select.countrylabel"/>
<strutshtml:select property="country" >
  <option value="">&nbsp;</option>
  <strutshtml:options property="countryList" />
</strutshtml:select>
```

The **option** tag provides a blank line in the selection list. The **options** tag refers to the `countryList` collection, which contains data retrieved by the `SelectForm` class.

5. Find the tags for the **submit** button.

```
<strutshtml:submit>
  <strutsbean:message key="button.submitall"/>
</strutshtml:submit>
```

The button has no attributes. The message tag supplies a custom label.

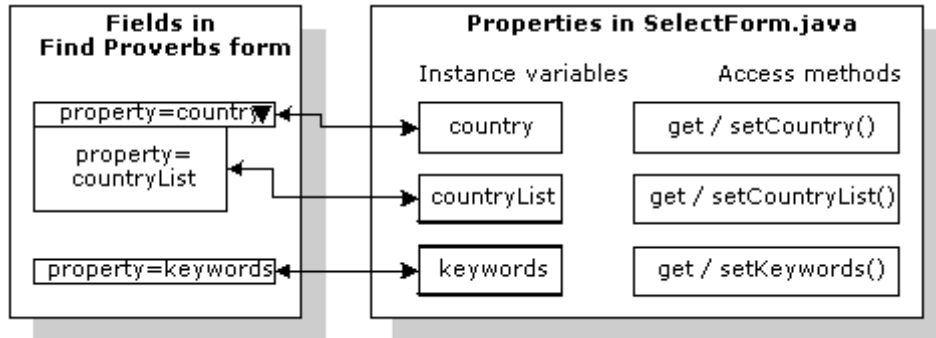
6. Close the file.

In the next exercise you'll see how an `ActionForm` class defines properties that match the fields in the form.

Supporting the form with an `ActionForm` class

When you use Struts, each form has a class that extends `ActionForm`. The class has properties that match the fields in the form. Your Action class has code that instantiates the form and accesses submitted data by getting property values.

The `SelectForm` class in the Proverbs application has properties for the country and keywords input fields. It also has a `countryList` property for the countries displayed in the dropdown list. The `reset()` method, which initializes and resets the form, calls the private `getList()` method, which queries the database for the list of countries.



EXERCISE 7-3: Examine the `SelectForm` class

In this exercise you will look at how the form implements properties and initializes data.

1. In the Workbench Navigation Pane, find the file `SelectForm.java` and open it.
2. In the Edit Pane, notice that the `SelectForm` class extends the Struts `ActionForm` class.
3. Find the properties that match the Struts form elements in the JSP page. The properties have:
 - Instance variable declarations
 - Get and set methods
4. Find the `reset()` method, which clears the input fields and retrieves the country list.
5. (Optional) Examine the `getList()` method.

The `getList()` method uses the same data access techniques that the today action uses. If you want, review these techniques in EXERCISE 6-1: “Retrieve data from the database” and EXERCISE 6-3: “Use an Action class to set up the data for a JSP page”.

6. Close the file.

Processing for the actions

As you know, each Struts action uses a class that extends Action. In the Find Proverbs activity, both of its actions use the same Action class.

In the code you can identify the current action by checking the path property of the ActionMapping object passed to perform(). In the Proverbs application the SelectAction class doesn't check the path property; instead it checks whether the fields in the form have values:

- If the form fields are empty, perform() returns the **failure** keyword; for the select action, the empty form is displayed, and for the results action, the no-results page is displayed.
- If the form has values, it must be the results action; so perform() retrieves the proverb data and returns the forward keyword **success**. The results page displays the data.



EXERCISE 7-4: Examine the SelectAction class

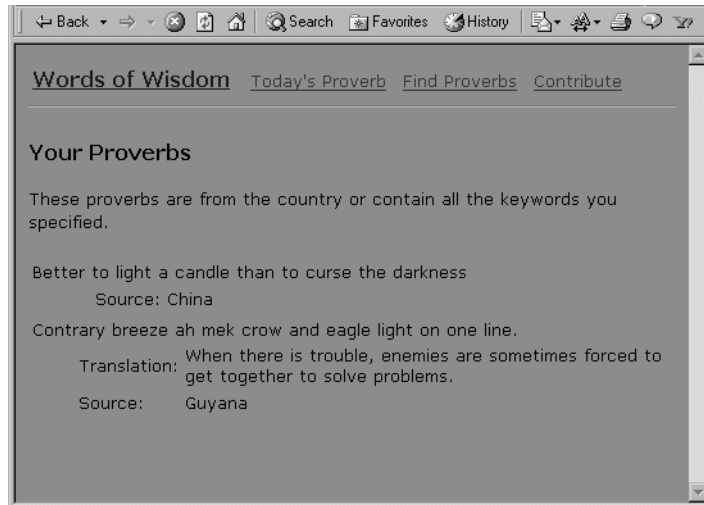
In this exercise you will look at code that instantiates a SelectForm object and retrieves a collection of data.

1. In the Workbench Navigation Pane, find the file **SelectAction.java** and open it.
2. In the Edit Pane, notice that the SelectAction class extends the Struts **Action** class.
3. Look at the code for:
 - Instantiating a **SelectForm** object if it doesn't exist—in the code **if (form == null)**
 - Getting **country** and **keyword** data the user entered by calling the form's get methods
 - Retrieving proverb data based on the user's criteria
 - Saving the retrieved proverb collection in the request
 - Returning a **forward** keyword to the controller
4. Close the file.

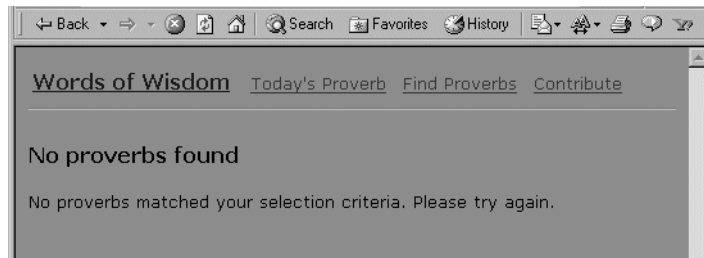
Displaying the retrieved data

The **results** action uses the same Action class as **select**, but the configuration specifies different JSP pages for the forward keywords. There are two pages:

- The **success** page displays all the proverbs that match the search criteria. As described for the Today's Proverb activity in "Retrieving data in the Action class" on page 92, SelectAction saves a collection of Proverb objects in the request, which the JSP page can access.



- The **failure** page reports that no proverbs matched the criteria. The text of the message is stored in the ApplicationResources file.



EXERCISE 7-5: Examine the JSP pages that show the results of the search

In this exercise you will see code for displaying the retrieved proverbs as well as the page for reporting no results.

1. Open **selectResults.jsp**.
2. Find the Struts **iterate** tag and look at the tags that display the properties of the Proverb object.

Do you remember how the data tags work? If not, review “Struts tags for displaying data” on page 93.

3. Open **selectFailed.jsp**.

4. Find the message key **page.selectfailed.info**, which specifies the text reporting that no proverbs were found.
5. (Optional) Open **ApplicationResources.properties** and read the actual text for that message key.
6. Close the files.

Deploying and testing the form

Now you are ready to test the Find Proverbs activity.



EXERCISE 7-6: Deploy the application

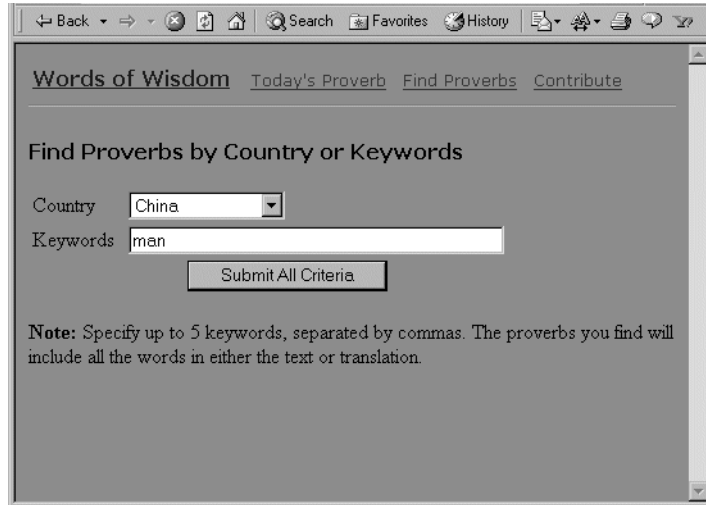
1. In Workbench, select **Project>Build and Archive**.
There should be no build errors in the Output Pane.
2. Select **Project>Deploy Archive** from the menu.
There should be no deploy errors in the Output Pane.



EXERCISE 7-7: Test the Find Proverbs activity

1. If your browser is running, open a new browser window. If not, start your browser.
2. Enter the URL for the application. A typical URL might be:
`http://localhost/ProverbsCloud/ProverbStart/index.jsp`
3. Click **Find Proverbs** in the application's navigation menu.
You see the Find Proverbs form.
4. Fill in the form. You can select a country from the dropdown list, specify keywords, or both.

TIP Try the country **China** and the keyword **man**. Try the keyword **light** with **Country** left blank.

A screenshot of a web browser window. The browser's address bar shows the URL 'Words of Wisdom'. The page title is 'Words of Wisdom' and there are navigation links for 'Today's Proverb', 'Find Proverbs', and 'Contribute'. The main content area is titled 'Find Proverbs by Country or Keywords'. It contains a form with a 'Country' dropdown menu set to 'China' and a 'Keywords' text input field containing 'man'. Below the input fields is a 'Submit All Criteria' button. A note below the form states: 'Note: Specify up to 5 keywords, separated by commas. The proverbs you find will include all the words in either the text or translation.'

If the application finds proverbs that meet your criteria, it displays **selectResults.jsp**, titled Your Proverbs.

If no proverb meets the criteria, it displays **selectFailed.jsp**.

Summary of what you've done

Developing the application In this lesson you built or learned about these parts of the Proverbs tutorial application:

- Action elements specifying how the controller handles displaying the form and the results
- JSP page with a form for the user's search criteria
- ActionForm class for the HTML form
- Action class that sets up and processes either action
- JSP pages for displaying results or reporting failure

Using Workbench tools You used these tools in Workbench:

- Edit Pane
- Building and archiving (Project>Build and Archive)
- Deployment (Project>Deploy Archive)

Next lesson In the next lesson you will learn about processing a form that updates the database.

8

Defining a Form for Database Update

What you will learn

In this lesson you will reinforce what you've already learned about Struts forms and actions. You'll also look at code and configure the actions that let the user update the database.

You will learn about:

- Configuring actions for contributing a proverb
- The classes that support the contribute actions
- Deploying and testing the finished application

What you will do

1. Define the contribute actions in the Struts configuration file
2. Examine the code for the contribute actions
3. Deploy the application
4. Test the contribute action and the rest of the application's activities

How long will it take? About 10 minutes

NOTE In this lesson you will need to run your J2EE application server for the deployment exercise.

Configuring actions for contributing a proverb

The Contribute a Proverb activity is similar to Find Proverbs. The contribute action handles displaying the form, and the saveResults action handles the submitted data.

The new wrinkle in this activity is that each action has its own Action class. The ContributeAction class instantiates the form if necessary. The SaveResultsAction class gets the submitted data and constructs the SQL for updating the database.



EXERCISE 8-1: Define the contribute actions in the Struts configuration file

In this exercise you will add two actions to the configuration file.

NOTE You can copy the XML for this exercise from the file **CutAndPasteCode.txt** in the **Workbench-install-dir/docs/tutorial/TutorialFiles/proverbs** directory.

1. In the Workbench Navigation Pane, find and open **struts-config.xml**.
2. Inside the action-mappings element after the closing tag **</action>** of the results action, enter this XML for the action that displays the contribute form:

```
<action path="/contribute"
        type="com.proverb.ContributeAction"
        name="contributeForm"
        validate="false">
  <forward name="success" path="/contribute.jsp"/>
  <forward name="failure" path="/contribute.jsp"/>
  <forward name="cancel" path="/index.jsp"/>
</action>
```

3. Still inside the action-mappings element after **</action>**, enter this XML for the action that updates the database:

```
<action path="/saveProverb"
        type="com.proverb.SaveProverbAction"
        name="contributeForm"
        scope="session"
        input="/contribute.jsp"
        validate="true">
  <forward name="success" path="/contributeResult.jsp"/>
  <forward name="failure" path="/contributeFailed.jsp"/>
  <forward name="cancel" path="/index.jsp"/>
</action>
```

4. In the **form-beans** section at the beginning of the file, look at the **form-bean** element that defines **contributeForm** and identifies its Java class.

```
<form-bean name="contributeForm"
           type="com.proverb.ContributeForm"/>
```

5. Save and close the file.

The classes that support the contribute actions

You've seen that there are two actions for the Contribute a Proverb activity. Both actions have their support classes:

Action	Supporting classes and JSP pages
contribute	ContributeAction.java ContributeForm.java contribute.jsp
saveProverb	SaveProverbAction.java ContributeForm.java contributeResult.jsp, contributeFailed.jsp

The Contribute a Proverb activity is implemented a little differently from Find Proverbs:

- Each action has its own Action class, which makes the code in each one simpler
- The ContributeForm uses validation; the class has a validate() method, and validation is turned on in the configuration file. If validation fails, the controller displays contribute.jsp again with error messages. The application requires the user to specify the proverb text and a country; it does not require a translation.



EXERCISE 8-2: Examine the code for the contribute actions

In this exercise you will look at how the contribute action differs from the Find Proverbs code described in Lesson 7, “Defining a Form and Results Page”.

1. In Workbench, open **ContributeForm.java** and find the **validate()** method.
The method builds an ActionErrors object. Like the rest of the application, the text for the error messages is in ApplicationResources.properties and the code refers to the message keys.
2. Open **contribute.jsp** and find the **errors** custom tag.
When validation fails, the tag includes the error text in the page.
3. Open **ContributeAction.java** and note the reduced amount of code.
All this class needs to do is instantiate the form.

4. Open **SaveProverbAction.java** and look at the code that processes the submitted proverb data.

The code constructs a Proverb object of the submitted data and calls a method in ProverbDataAccess that updates the database.

```
Proverb pvb = new Proverb(
    contribform.getProverb(),
    contribform.getTranslation(),
    contribform.getCountry() );
int result = pda.insertIntoProverbTable( pvb );
```

The code that gets a DataSource and instantiates the ProverbDataAccess object should look familiar.

When the update succeeds, the form is removed from the session so the data won't be processed again.

```
session.removeAttribute(mapping.getAttribute());
```

5. Close the files.

Deploying and testing the finished application

You've reached the end of this application, and it's time to test all the activities.



EXERCISE 8-3: Deploy the application

These are the same deployment instructions you've used before.

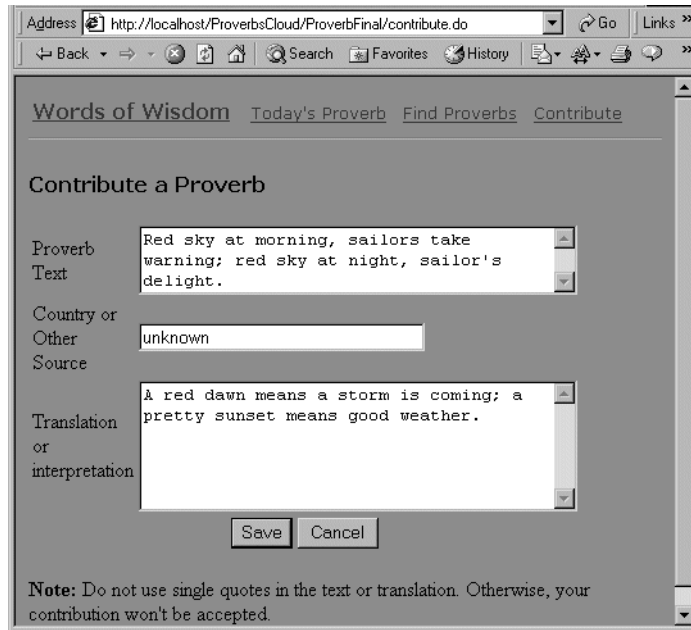
1. In Workbench, select **Project>Build and Archive**.
There should be no build errors in the Output Pane.
2. Select **Project>Deploy Archive** from the menu.
There should be no deploy errors in the Output Pane.



EXERCISE 8-4: Test the contribute action and the rest of the application's activities

1. Open a new browser window.
2. Enter the URL for the application. A typical URL might be:
`http://localhost/ProverbsCloud/proverbs/index.jsp`

3. Click **Contribute** in the application's navigation menu.
You see the contribute form.
4. Fill in the form. You can leave translation blank, but the proverb text and source are required.



Address <http://localhost/ProverbsCloud/ProverbFinal/contribute.do> Go Links »

← Back → Search Favorites History »

[Words of Wisdom](#) [Today's Proverb](#) [Find Proverbs](#) [Contribute](#)

Contribute a Proverb

Proverb Text: Red sky at morning, sailors take warning; red sky at night, sailor's delight.

Country or Other Source: unknown

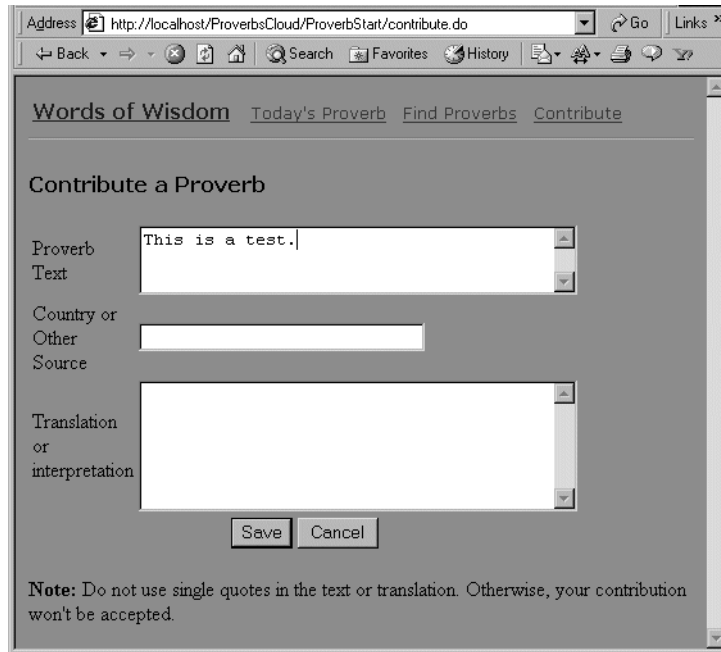
Translation or interpretation: A red dawn means a storm is coming; a pretty sunset means good weather.

Save Cancel

Note: Do not use single quotes in the text or translation. Otherwise, your contribution won't be accepted.

5. Click **Submit**.
If the application successfully updates the database with your proverb, it displays **contributeResult.jsp**. If updating the database fails, it displays **contributeFailed.jsp**.

6. Submit another proverb. Test the validation by leaving some of the fields blank.
You'll see a message reminding you what fields you left out. You can tell by looking at the URL that the application is in the second action of the Contribute Proverbs activity.



The screenshot shows a web browser window with the address bar displaying `http://localhost/ProverbsCloud/ProverbStart/contribute.do`. The page title is "Words of Wisdom" and the navigation menu includes "Today's Proverb", "Find Proverbs", and "Contribute". The main heading is "Contribute a Proverb". The form contains three text input fields: "Proverb Text" (with the value "This is a test."), "Country or Other Source", and "Translation or interpretation". Below the form are "Save" and "Cancel" buttons. A note at the bottom reads: "Note: Do not use single quotes in the text or translation. Otherwise, your contribution won't be accepted."

7. Test the other activities in the application.
8. When you're finished testing, close the browser.
You're done!

Summary of what you've done

Developing the application In this lesson you built or learned about these parts of the Proverbs tutorial application:

- Action elements specifying how the controller handles displaying the form and processing the submitted data
- JSP page with a form for the user's proverb data
- ActionForm class for the HTML form
- Action class for displaying the form

- Action class for processing the submitted form and updating the database
- JSP pages for displaying success or failure

Using Workbench tools You used these tools in Workbench:

- Edit Pane
- Building and archiving (Project>Build and Archive)
- Deployment (Project>Deploy Archive)

What's next Congratulations. You've finished building the Proverbs Web application based on the MVC architecture and Struts.

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

Index

A

archives

- about (tutorial) 33
- building (tutorial) 55
- structure of (tutorial) 35

C

classpaths

- for project (tutorial) 44

Cloudscape

- DBMS, obtaining 25
- tutorial database 24
- tutorial database with SilverStream 25
- tutorial database with WebLogic 28

compiling code (tutorial) 55

D

databases

- accessing from Web application (tutorial) 77
- choices (tutorial) 24
- identifying on server (tutorial) 82
- querying from application (tutorial) 88
- resource references (tutorial) 78
- retrieving data, classes (tutorial) 92
- updating (tutorial) 107

DataSource

- getting from naming service (tutorial) 84

deployment descriptors

- editing (tutorial) 57
- environment variables (tutorial) 78
- resource reference for database (tutorial) 78
- servlet mapping (tutorial) 62
- welcome page (tutorial) 63

deployment plans

- databases (tutorial) 82

E

environment variables

- in deployment descriptor (tutorial) 78

I

internationalization, Struts (tutorial) 13

J

J2EE application servers

- connection pool (tutorial) 82

J2EE development (tutorial) 3

JavaServer Pages

- JSP Wizard (tutorial) 50

JNDI

- getting DataSource (tutorial) 84

L

localization, Struts (tutorial) 13

P

projects

- about (tutorial) 33
- adding content (tutorial) 39, 48
- building and archiving (tutorial) 55
- classpath (tutorial) 44
- creating (tutorial) 36

ProverbFinal project (tutorial) 6

Proverbs application (tutorial)

- about 6
- classes for contribute actions 109
- classes for form 100
- classes for retrieving data 92
- Contribute a Proverb activity 107
- contribute action 107
- data source 24

- database setup 23
- database tables 19
- deploying 64
- deployment descriptor 57
- displaying data 93
- Find Proverbs activity 98
- form for selecting data 98
- form tags for selecting data 99
- JNDI and DataSource 84
- querying database 88
- results action 98
- saveResults action 107
- select action 98
- testing Contribute a Proverb activity 110
- testing Find Proverbs activity 104
- testing Today's Proverb activity 94
- testing welcome page 69
- Today's Proverb action 88
- updating database 107

S

- servlets
 - configuring (tutorial) 57
 - initialization parameters (tutorial) 60
- SilverStream eXtend Application Server
 - trial version, obtaining 25
- Struts (tutorial)
 - action implementation 89
 - actions and URL patterns 16
 - classes for form 100
 - configuring action 98
 - controller implementation 16
 - controller, configuring actions 90
 - displaying data 93
 - form tags 99
 - forms 15
 - framework 4
 - internationalization 13
 - model implementation 19
 - MVC architecture 4
 - planning tool for configuring actions 16
 - view implementation 8

T

- tutorials
 - developing a Web application (WAR) 3

U

- URLs
 - testing Proverbs application (tutorial) 69

W

- Web applications
 - Struts (tutorial) 4
- web.xml
 - editing (tutorial) 57