Pro Eclipse JST

Plug-ins for J2EE Development

Christopher M. Judd
and Hakeem Shittu

Apress®

Pro Eclipse JST: Plug-ins for J2EE Development

ISBN (pbk): 1-59059-493-2

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Steve Anglin
Technical Reviewer: Ben Houston
Editorial Board: Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Tony Davis, Jason Gilmore, Jonathan Hassell, Chris Mills, Dominic Shakeshaft, Jim Sumser
Associate Publisher: Grace Wong
Project Manager: Kylie Johnston
Copy Edit Manager: Nicole LeClerc
Copy Editor: Ami Knox
Assistant Production Director: Kari Brooks-Copony
Production Editor: Kelly Winquist
Compositor: Kinetic Publishing Services, LLC
Proofreader: April Eddy
Indexer: Michael Brinkman
Artist: Kinetic Publishing Services, LLC
Cover Designer: Kurt Krames
Manufacturing Manager: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

The source code for this book is available to readers at `http://www.apress.com` in the Downloads section.

CHAPTER 4

Introduction to JST

The J2EE Standard Tools (JST) project is an effort by the Eclipse development team to provide users of the Eclipse Platform a standardized framework for the creation of tools for the development of enterprise applications based on the J2EE Specification. The JST project together with the Web Standard Tools (WST) project, discussed in the next chapter, make up the Web Tools Platform (WTP) and jointly provide developers an arsenal of tools produced from the frameworks created by the respective projects. These tools are useful for the development, testing, and management of Java-based enterprise applications. The JST and WST projects are closely related because Java-based enterprise applications often rely heavily on non-Java technologies for core functionalities. The editing of XML-based deployment descriptors, which play an important part in the deployment of J2EE components, is one such example of this close relationship.

The development tools supplied by the project provide users of the ubiquitous Eclipse Platform with a cohesive set of plug-ins integrated into the Eclipse development environment to allow developers to create, debug, test, and deploy multitiered J2EE applications. Some of the tools are provided for use in the creation and maintenance of J2EE source artifacts such as JavaServer Pages (JSP), servlets, EJBs, and other deployable assets. Other tools are meant for activities that provide support to the development process such as the packaging of artifacts into deployable modules, the exploring of available Web Services, and also the management of application servers.

JST limits its scope to providing support for J2EE technologies even though, as we have earlier indicated, many non-Java technologies are often found within an enterprise application. Support for many of these non-Java technologies falls within the scope of the WST project. Expectedly, not all enterprise Java technologies are supported by JST. Support is provided for the standards that comprise the J2EE 1.4 Specification as well as a few other JCP-approved standards. XDoclet, a popular technology for annotating source artifacts, is also supported, even though it is not a JCP standard. Other popular enterprise Java technologies such as Hibernate and Velocity, which are not based on JCP standards, are, however, unsupported.

It is important to bear in mind that JST goes significantly beyond the provision of tools for enterprise development and additionally provides a framework from which further tools of usefulness to Java enterprise development can be created. A brief example of this is that Struts, the popular web development framework which is not supported by JST, could have support provided for it by tool extenders using the framework provided by JST. This way, the new plug-in seamlessly coexists with those provided by JST.

We begin this chapter by describing the scope of the JST project; we will also explore the benefits provided by this project. Discussion of the foundation provided by JST for the creation of J2EE-focused tools will follow, and we will conclude the chapter with an extensive overview of the tools provided by JST. We will be demonstrating the use of these tools more comprehensively throughout the entire book.

JST Scope

JST provides Eclipse with a comprehensive suite of tools for the development of Java-based enterprise applications conforming to the J2EE 1.4 Specification. Due to J2EE support being its core purpose, JST limits its scope to Java-based technologies. Many of the supported technologies, however, such as JavaServer Pages, have a dependence on open standards such as HTML and CSS. In these cases, the open standards fall within the scope of the WST project.

JCP standards, which lie at the core of the J2EE Specification, feature heavily in the list of supported technologies, though some newer JCP standards that are not a part of the J2EE 1.4 Specification are not supported. Notable items in this list include JDO (JSR 243—Java Data Objects 2.0), which defines Java object persistence, and JSF (JSR 252—JavaServer Faces), which provides standard APIs and tag libraries for web interface development.

Additionally, many non-JCP technologies are beyond the scope of JST, including such popular frameworks as Struts, Velocity, and XMLC. XDoclet, as we noted earlier, is supported, however, largely due to the absence of a JCP standard for J2EE annotation.

We present in Table 4-1 a list of supported technologies in JST and the JSR that defines each technology where applicable. You can view any of these specifications by visiting the Java Community Process website at `http://www.jcp.org`.

Table 4-1. *JST-supported Technologies by JCP Standards*

JCP Standard	Title
JSR 3	Java Management Extensions (JMX) 1.2
JSR 5	Java API for XML Parsing (JAXP) 1.2
JSR 45	Debugging Support
JSR 54	JDBC API 3.0
JSR 67	SOAP with Attachments API for Java (SAAJ) 1.2
JSR 77	J2EE Management API 1.0
JSR 88	Deployment API 1.1
JSR 93	Java API for XML Registries (JAXR) 1.0
JSR 101	Java API for XML-based RPC (JAX-RPC) 1.1
JSR 109	Web Services
JSR 112	J2EE Connector Architecture (JCA) 1.5
JSR 115	Java Authorization Contract for Containers (JACC)
JSR 152	JavaServer Pages (JSP) 2.0
JSR 153	Enterprise JavaBeans (EJB) 2.1
JSR 154	Servlets 2.4
JSR 907	Java Transaction API (JTA) 1.0
JSR 914	Java Message Service (JMS) 1.1
JSR 919	JavaMail 1.3

In addition to these JCP standards, the following non-JCP standards and technologies are also within the scope of JST:

- Java Authentication and Authorization Service (JAAS)
- Java Naming and Directory Interface (JNDI)
- XDoclet

■ **Note** XDoclet is supported by JST as a vehicle for J2EE annotation pending the creation of a JSR for this important purpose. An expanded discussion of the role, configuration, and usage of XDoclet in JST is provided in Chapter 8.

JST Goals

A major goal of JST is to extend the Eclipse Platform into the enterprise software development space in the hopes of expanding usage of the platform. Similar to how the Java Development Tools (JDT) provided Eclipse with the necessary tools for the creation and management of standard Java components, and was thus rewarded with a major influx of users who adopted the platform, so does JST expect to generate an influx of new users to adopt the platform.

Although third-party plug-ins exist that provide Eclipse with support for the creation of J2EE applications, there remains some associated difficulty with the process of finding and using these plug-ins for a project. Chief among these difficulties is the need for developers to individually seek and assemble the right collection of commercial and open source plug-ins that suit the needs of their project. A developer could eventually find a suitable assembly of plug-ins to use for a specific project, but the lack of a standard set of tools remained a significant source of complaint due to the time—and sometimes cost—intensive nature of the searching for and assembling of components. Sites like Eclipse Plugin Central (`http://www.eclipseplugincentral.com/`) try to make the process of assembly easier by providing users with a repository of different plug-ins as well as information about these plug-ins and tutorials on how to use them. This service is, however, only available to paid subscribers.

It is reasonable to expect that users of Eclipse would want a readily available set of tools immediately accessible to them for their development needs. This is especially prevalent among users new to Eclipse and probably more familiar with IDEs that provide readily available development tools. In fact, this requirement of assembling plug-ins for developing J2EE applications is seen as a deterrent to the adoption of the platform for many new users. The JST project intends to remove this barrier by providing the necessary development tools that can be used for J2EE development.

It should be noted that support for third-party plug-ins remains a fundamental part of the Eclipse architecture, and the Web Tools Project was created to enhance rather than diminish this purpose. In fact, beyond providing a standardized set of development tools, the JST project importantly provides a framework for the development of additional tools for the enterprise Java community. This framework, known as the *J2EE Core Model* (JCM), is provided to allow the development of tools that would seamlessly integrate with existing JST tools and the Eclipse Platform. It is hoped that this would encourage vendors to extend support for their proprietary tools to the Eclipse Platform and also for other companies and groups to develop plug-ins for Eclipse to support whatever technology or standard the platform might not currently be providing support for.

The aggregate goal of the JST project is a continued expansion of the user base of the platform, and this is a welcome proposition as it helps to strengthen the platform as an invaluable tool in the software development process. The JST project hopes the release of its toolkit will bring about an inflow of new users to the platform and also swell the ranks of J2EE developers. This growth is expected to in turn encourage vendors to continue to provide the platform with support for their commercial tools, and also encourage open source developers to provide plug-ins that support newer technologies that developers might become interested in.

J2EE Core Model

We have mentioned earlier that JST provides a foundation for the development of J2EE-focused tools. This foundation is known as the J2EE Core Model (JCM), and it comprises frameworks and object models that abstract out core functionalities of J2EE artifacts and components. It additionally provides APIs for accessing and manipulating these functionalities. The JCM is made available to third-party developers to extend the number of tools available for the Eclipse Platform and importantly provide support for additional technologies not currently supported.

It should be understood that the JCM is not provided for use in the creation of J2EE applications, but instead as a base infrastructure for creating J2EE development tools. The commitment of Eclipse to the extension of the platform is thus apparent. The tools provided by the JST project are actually themselves an extension of the JCM. We will discuss these tools at length later in the chapter, but we must first examine the models that are provided by the JCM, which are listed here:

- **J2EE Project Model**: This provides the Eclipse Platform with the framework for managing a J2EE project. The project model supports a flexible structure whereby a project contains a collection of artifacts that constitute either a complete J2EE application or deployment modules that represent a portion of such an application. It also provides a mechanism for the management of build activities and deployment of created artifacts.

- **J2EE Editor Model**: This model extends the standard Eclipse editor to provide support for the creation and editing of J2EE source artifacts such as JSPs, servlets, EJBs, etc. It provides a base for the creation of various text and graphical editors, giving support for such essential editing functionalities as syntax coloring, code assist, refactoring, and quick fixes.

- **J2EE Artifacts Model**: This model represents J2EE source and deployment artifacts like JSPs, EJBs, deployment descriptors, etc., that can be created and managed within a project together with other artifacts such as image resources, text files, and various other files that may be packaged into a deployable module. The model further represents the deployment modules in their archival state, i.e., WAR, EJB JAR, EAR, and RAR files. Natures, builders, validators, and EMF models associated with a project will also be represented by this model.

- **J2EE Server Model**: The server model provides the abstractions required to support the deployment of modules to many different types of application servers. It additionally provides a unified control mechanism to start, administer, and stop these J2EE application servers. The model also provides the means for managing the configuration details of these servers including environment parameters, JVM parameters, and classpaths.

J2EE module management such as packaging, deployment, debugging, removal, import, and export of the J2EE modules are managed by the model.

J2EE Standard Tools

We will now examine the tools resulting from the JST project. As stated earlier, these tools are of use in creating J2EE source artifacts or providing supporting functions for the creation of J2EE applications. The tools are an exemplar implementation of the JCM described previously and were created by extending the models and the APIs made available by the JCM. The resulting product of this activity is an integrated set of tools, views, perspectives, and wizards useful for various J2EE development activities.

JST provides several tools for enterprise application development, with each tool targeted at aiding in a specific domain of activities related to developing a J2EE application. Servlet Tools for instance are useful for creating servlet components that are deployed within the web container of an application server, while EJB Tools are used to create EJBs that are deployed within the EJB container. We discuss the tools of the JST in this section.

J2EE Project and Module Tools

The J2EE Project Tools support the creation of a J2EE project as a collection of source artifacts that each constitute a deployment module of a J2EE application. The project also contains important configuration information that is stored and managed within the project workspace but excluded from the deployable module resulting from the project. These tools support the creation of five types of projects:

- **EJB Project**: Consists of Enterprise JavaBean components and supports source artifacts that would be compiled and deployed within the EJB container of an application server.
- **Dynamic Web Project**: Consists of servlets, JSPs, and additional Web Tier components such as taglibs, HTML documents, images, etc. that are targeted for deployment inside the web container of an application server.
- **Application Client Project**: Consists of application client components often designed for the consumption of services provided by an enterprise application.
- **Connector Project**: Consists of source files for the creation of connector applications to integrate legacy systems with J2EE applications as specified by JSR 112 (J2EE Connector Architecture 1.5).
- **Enterprise Application Project**: Consists of modules that represent a complete enterprise application. This project provides the ability to reference several EJB, Web, Application Client, and Connector Projects whose products will be deployed together.

The J2EE Module Tools additionally provide multiple useful features in the creation of a J2EE project. These features include the ability to create and manage artifacts within each of the J2EE projects described earlier and also provide a module structure that represents the deployable artifacts within the project. The module structure can then be packaged into a suitable J2EE deployable archive.

Figure 4-1 shows a dialog box of the Project Creation wizard and displays some of the valid choices for an Enterprise Application Project. Additional dialog boxes are provided to developers, allowing them to include further details specific to the project type chosen on this initial project creation page. Figure 4-2 shows the screen provided when an Enterprise Application Project (which is highlighted in Figure 4-1) is being created.

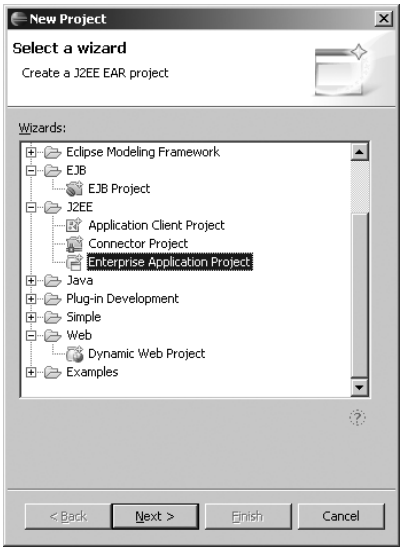

Figure 4-1. *Project Creation wizard*

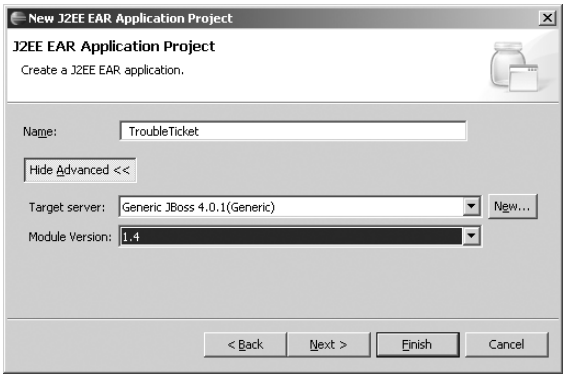

Figure 4-2. *Enterprise Application Project dialog box*

The project tool provides support for validating the deployment module within projects, editing its deployment descriptor, as well as packaging the contents within the module. The tool also provides the ability to create projects through the importation of modules. We will provide a more in-depth discussion of project structures in Chapter 7.

J2EE Server Tools

J2EE applications need to be deployed into a web or EJB container as we discussed in Chapter 1, and these containers exist within a J2EE application server. In JST, management of servers to which J2EE applications would be deployed is handled by the J2EE Server Tools. The Server Tools provide a mechanism for the definition of server runtime environments and the creation of instances of an application server from this definition. They also ensure that when a project of a particular type is being created, it can be deployable to the server chosen for the project. Tomcat, for instance, being an application server consisting only of a web container, would not be available as a choice for the creation of an EJB Project. Table 4-2 shows the application servers that are supported as well as the project type and version that they support.

Table 4-2. *Project Types and Supported Application Servers*

Web Version	EJB Version	Application Server
2.2, 2.3, and 2.4	1.1, 2.0, and 2.1	Apache Geronimo 1.0
2.2		Apache Tomcat version 3.2
2.2 and 2.3		Apache Tomcat version 4.0
2.2 and 2.3		Apache Tomcat version 4.1
2.2, 2.3, and 2.4		Apache Tomcat version 5.0
2.2, 2.3, and 2.4		Apache Tomcat version 5.5
2.2, 2.3, and 2.4	1.1, 2.0, and 2.1	BEA WebLogic Server version 8.1
2.2, 2.3, and 2.4	1.1, 2.0, and 2.1	BEA WebLogic Server version 9.0
2.2, 2.3, and 2.4	1.1, 2.0, and 2.1	IBM Websphere 6.0.x
2.2, 2.3, and 2.4	1.1, 2.0, and 2.1	JBOSS 3.2.3
2.2, 2.3, and 2.4	1.1, 2.0, and 2.1	JonAS 4.1.4

The tool is also capable of managing the definition of multiple server runtime environments, thereby allowing deployable modules from different J2EE projects within the workbench to be deployed to their own specific target deployment servers. Figure 4-3 shows an example of the Installed Server Runtime Environments Preferences page where definitions of J2EE server runtime environments can be configured.

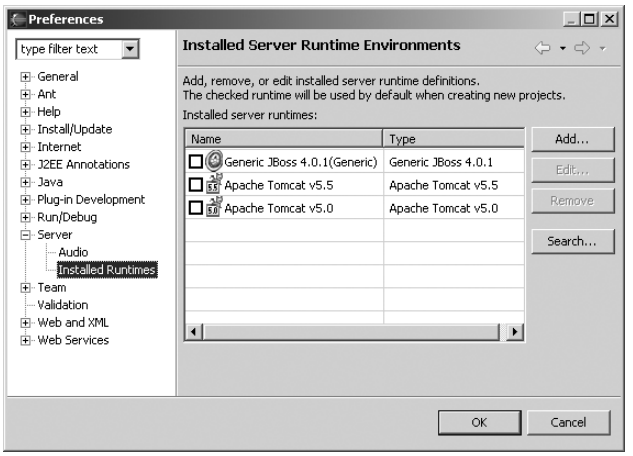

Figure 4-3. *Server runtime definition*

Additional configuration is available for each defined server runtime environment, allowing you to provide such items as environment variables and classpaths for the runtime instance, as well as modify the default startup and VM arguments of the server. J2EE Server Tools additionally provide a mechanism to create instances of servers from these definitions and control execution of the server runtime instances through the Servers View. The Servers View provides the ability to start, stop, debug, and profile each application server. Output generated from each of these processes is written to the Console View.

■ **Tip** The Servers View is available as a part of the J2EE Perspective, discussed in its own section later in this chapter. It can also be accessed by selecting Window ➤ Show View ➤ Other ➤ Server ➤ Servers. The Console View, which is not a part of the J2EE Perspective, can be accessed by selecting Window ➤ Show View ➤ Other ➤ Basic ➤ Console.

Servlet Tools

Servlets provide J2EE with the ability to dynamically process requests and provide responses to invocations that support a request/response model. They are most commonly used with HTTP, which expectedly supports the request/response model, and is therefore an important part of most J2EE web applications. The J2EE Servlet Tools include support for servlets in JST by providing such functions as wizard-assisted creation of servlet classes. It additionally automates the insertion of configuration information about each created servlet such as the `<servlet>` and `<servlet-mapping>` elements into the deployment descriptor of the web project that references the servlet. Figure 4-4 shows the Create Servlet wizard page available during the creation of a servlet, which prompts for the servlet name, description, initialization parameters, and URL mappings that are needed by the servlet.

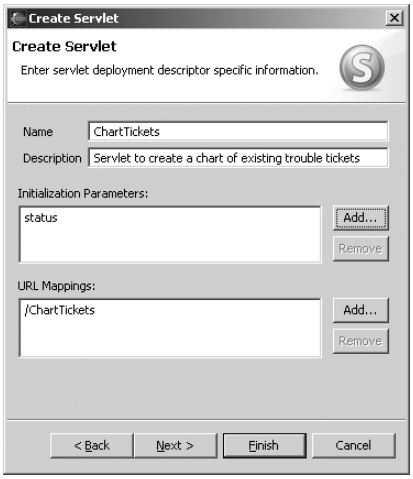

Figure 4-4. *Create Servlet wizard*

Additional wizard pages allow the user to specify the associated package for this servlet as well as choose access modifiers for the servlet class and method stubs that should be generated in the new servlet. Figure 4-5 displays one of these wizard pages showing the default modifiers, implemented interfaces, and method stub choices preselected by the tool. As with all wizards, these choices can easily be altered to suit the needs of your project.

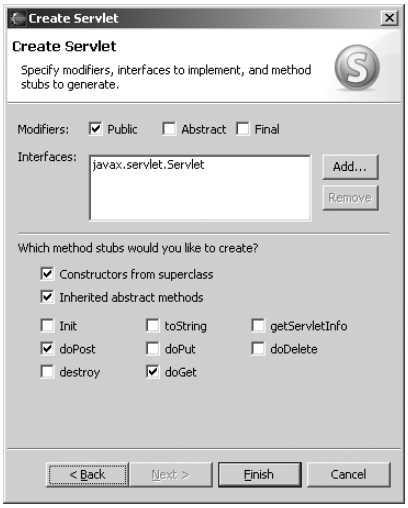

Figure 4-5. *Selection of methods to insert into servlet*

For each of the methods chosen from Figure 4-5, an appropriate method signature is inserted into the generated servlet source file with a `TODO` task tag statement included that denotes where

user-based implementation is required. The text associated with these task tags is immediately reflected in the Tasks View, providing a cohesive location for the developer to track uncompleted tasks. Listing 4-1 shows the servlet file generated by the wizard, including the XDoclet annotations that would be used to insert configuration details into the `web.xml` deployment descriptor.

Listing 4-1. *Generated Servlet File*

```
package com.projst.ticket.web;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class for Servlet: ChartTickets
 *
 * @web.servlet
 *   name="ChartTickets"
 *   display-name="ChartTickets"
 *   description="Servlet to create a chart of existing trouble tickets"
 *
 * @web.servlet-mapping
 *   url-pattern="/ChartTickets"
 *
 * @web.servlet-init-param
 *    name="status"
 *    value="open"
 *    description="Ticket type to include in chart"
 *
 */
 public class ChartTickets
           extends javax.servlet.http.HttpServlet
           implements javax.servlet.Servlet {
    /**
     *
     */

    /* (non-Java-doc)
     * @see javax.servlet.http.HttpServlet#HttpServlet()
     */
    public ChartTickets() {
        super();
    }
```

```
    /* (non-Java-doc)
     * @see HttpServlet#doGet(HttpServletRequest arg0, HttpServletResponse arg1)
     */
    protected void doGet(HttpServletRequest arg0, HttpServletResponse arg1)
                    throws ServletException, IOException   {
        // TODO Auto-generated method stub
    }

    /* (non-Java-doc)
     * @see HttpServlet#doPost(HttpServletRequest arg0, HttpServletResponse arg1)
     */
    protected void doPost(HttpServletRequest arg0, HttpServletResponse arg1)
                    throws ServletException, IOException   {
        // TODO Auto-generated method stub
    }
}
```

USING TASK TAGS

Eclipse provides *task tags* to serve as markers within documents being edited. When included, comments defined as tasks immediately show up in the Tasks View, providing you with information about additional tasks that need to be completed within your application. These tags can be defined within the Task Preferences page, which can be accessed by selecting Window ➤ Preferences ➤ Java ➤ Compiler ➤ Task Tags. Tags can then be added or edited on this Preferences page.

The `TODO` tag is the default task tag, and it additionally defaults to a priority of normal. To use the `TODO` tag, a line like the following is included within your source code file:

```
// TODO Auto-generated method stub
```

You will notice that the `TODO` task tag is inserted as a Java comment tag, ensuring that your code can be compiled in any IDE without causing errors, since comment tags are universally ignored. Figure 4-6 shows an example of the Tasks View when the servlet file shown in Listing 4-1 is being edited.

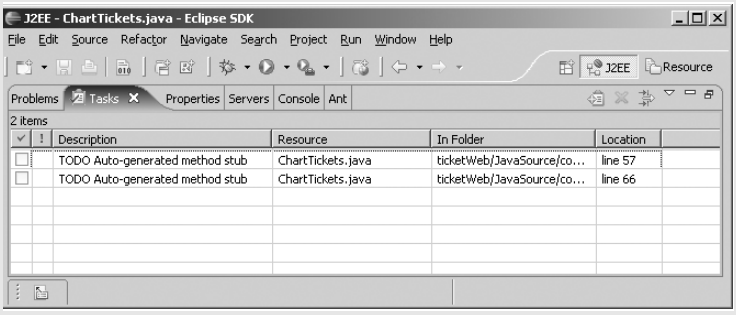

Figure 4-6. *Tasks View displaying task tags from the servlet in Listing 4-1*

You may notice that XDoclet tags are being used in Listing 4-1. These tags provide annotations that can easily be modified to manage information included within the `web.xml` deployment descriptor about this servlet. Listing 4-2 shows the values inserted into the `web.xml` file based on the content of the XDoclet annotations in Listing 4-1.

Listing 4-2. *Servlet Tag Generated into the web.xml Deployment Descriptor*

```
<servlet>
   <servlet-name>ChartTickets</servlet-name>
   <display-name>ChartTickets</display-name>
   <description>
      <![CDATA[Servlet to create a chart of existing trouble tickets]]>
   </description>
   <servlet-class>com.projst.ticket.web.ChartTickets</servlet-class>

   <init-param>
      <param-name>status</param-name>
      <param-value>open</param-value>
      <description><![CDATA[Ticket type to include in chart]]></description>
   </init-param>
</servlet>

<servlet-mapping>
   <servlet-name>ChartTickets</servlet-name>
   <url-pattern>/ChartTickets</url-pattern>
</servlet-mapping>
```

It should be noted that servlets are not edited with special editors, but instead with a standard Java editor that is used in creating a Java class. Due to the servlets' nature of being Java source files, creation of specialized editors for servlets is not required. In Chapter 13, we have provided an extended discussion of the use and creation of servlets with JST.

JSP Tools

JavaServer Pages are an integral part of the presentation layer of many enterprise applications, providing access to programmatic functions through a web interface. JST provides a mechanism for the creation and management of JavaServer Pages source files through the JSP Tools. The JSP Tools rely heavily on WST to provide a core foundation for its functions. This is because a JSP file can contain code based on HTML, JavaScript, and CSS technologies in combination with Java expressions and custom tags. Formatting and content assist for the technologies based on open standards are managed by WST, while JST manages the same for Java-related content within the page. For a JSP page, important features such as syntax coloring of the different elements within the JSP file like Java code, JSP tags, HTML tags, and JavaScript code is supported. Figure 4-7 shows the JavaServer Page wizard used for the initial creation of JSP documents.

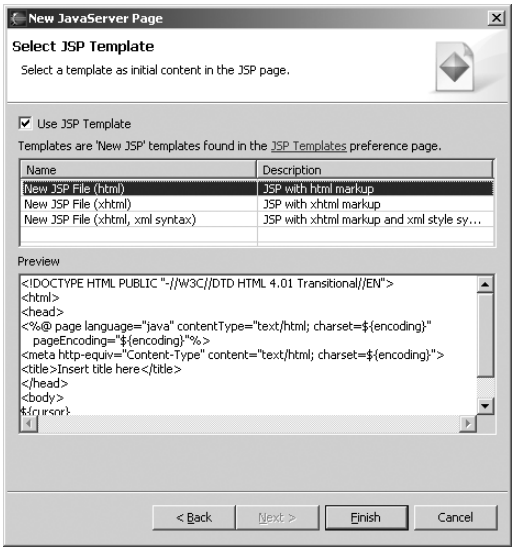

Figure 4-7. *JavaServer Page wizard*

Creation of JSPs is template based, and these templates can be managed through the JSP Templates Preferences page by selecting Window ➤ Preferences ➤ Web and XML ➤ JSP Files ➤ JSP Templates. We provide additional information about the creation and management of these templates in Chapter 12. Listing 4-3 shows the JSP source file generated from choosing one of the default templates shown in Figure 4-7 earlier.

Listing 4-3. *A Generated JSP File*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
<title>Insert title here</title>
</head>
<body>

</body>
</html>
```

JSP Tools also provide debugging support for JSP source artifacts as specified by JSR-45 with the ability to set breakpoints in the JSP editor as well as step through code execution.

EJB Tools

JST greatly simplifies the process of creating Enterprise JavaBeans through the inclusion of the EJB Tools. These tools provides wizard-assisted mechanisms for the creation of entity, session, and message-driven beans, thereby automating the process of creating these artifacts and greatly reducing the possibility of error that could occur if they are directly created by a developer. One such wizard, in this case a wizard page for the creation of a message-driven bean, is shown in Figure 4-8.

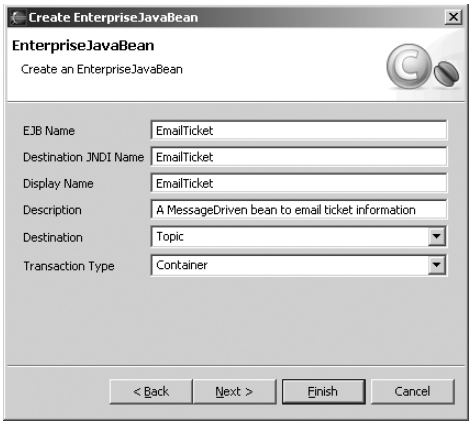

Figure 4-8. *EJB Creation wizard*

Beyond the provision of wizards, J2EE tools additionally provide annotation support for the source file generated in the form of XDoclet attributes. These attributes automate certain mechanical tasks that are part of the process of creating an EJB such as the creation of home and remote interfaces for the EJB or the inclusion of appropriate values for such properties as JNDI names and EJB type within the deployment descriptor of the module. Listing 4-4 shows the annotated source file generated by Eclipse from the wizard shown in Figure 4-8.

Listing 4-4. *A Generated Stateless Session Bean Including XDoclet Annotations*

```
/**
 *
 * <!-- begin-user-doc -->
 * A generated session bean
 * <!-- end-user-doc -->
 * *
 * <!-- begin-xdoclet-definition -->
 * @ejb.bean name="Facility"
 *       description="A session bean named Facility to manage information about the "
```

```
 *           display-name="Facility"
 *           jndi-name="Facility"
 *           type="Stateless"
 *           transaction-type="Container"
 *
 * <!-- end-xdoclet-definition -->
 * @generated
 */

public abstract class FacilityBean implements javax.ejb.SessionBean {

    /**
     *
     * <!-- begin-xdoclet-definition -->
     * @ejb.interface-method view-type="remote"
     * <!-- end-xdoclet-definition -->
     * @generated
     *
     * //TODO: Must provide implementation for bean method stub
     */
    public String foo(String param) {
        return null;
    }
}
```

We have provided extended descriptions of the J2EE EJB Tools in Chapter 8, Chapter 9, and Chapter 10.

Java Web Services Tools

JST provides the Java Web Services Tools, which allow users to automate the process of creating and consuming a Web Service. Built upon Apache Axis (`http://ws.apache.org/axis/`), a widely used platform that simplifies the process of creating and accessing Web Services, the tools provide wizard-assisted creation of Java Web Services from EJBs, JavaBeans, or WSDL files. It also provides wizards for the creation of Java clients that consume Web Services made available by others. Figure 4-9 shows the initial creation screen during the creation of a Web Service client.

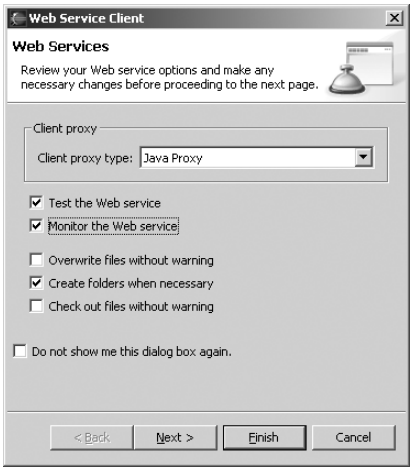

Figure 4-9. *Web Service Client Creation wizard*

The Web Services Tools integrates into the context menu available within the Project Explorer to provide a mechanism for the creation of Web Services whenever servlet or stateless session beans are selected within the Project Explorer. Facility to test the Web Service created is also provided by the tool through the automated generation of JSP pages that can invoke each selected Web Service method or by using the Web Services Explorer.

Web Services Explorer

A significant addition to Eclipse is the inclusion of the Web Services Explorer as part of the Web Services Tools. This tool, which is available from the Run ➤ Launch Web Services Explorer menu selection, allows you to browse UDDI registries for available Web Services that can then be used within you application. It provides a facility to query these registries for available Web Services and additionally automates the generation of clients to connect to any available Web Service. Figure 4-10 shows a sample of the Web Services Explorer execution.

The Web Services Explorer tool provides the additional benefit of being able to test any Web Service that is available from a UDDI registry providing a useful evaluation tool to test out the functioning of a Web Service before referencing it within an application. This tool can also test Web Services that exist within the workbench in a similar fashion.

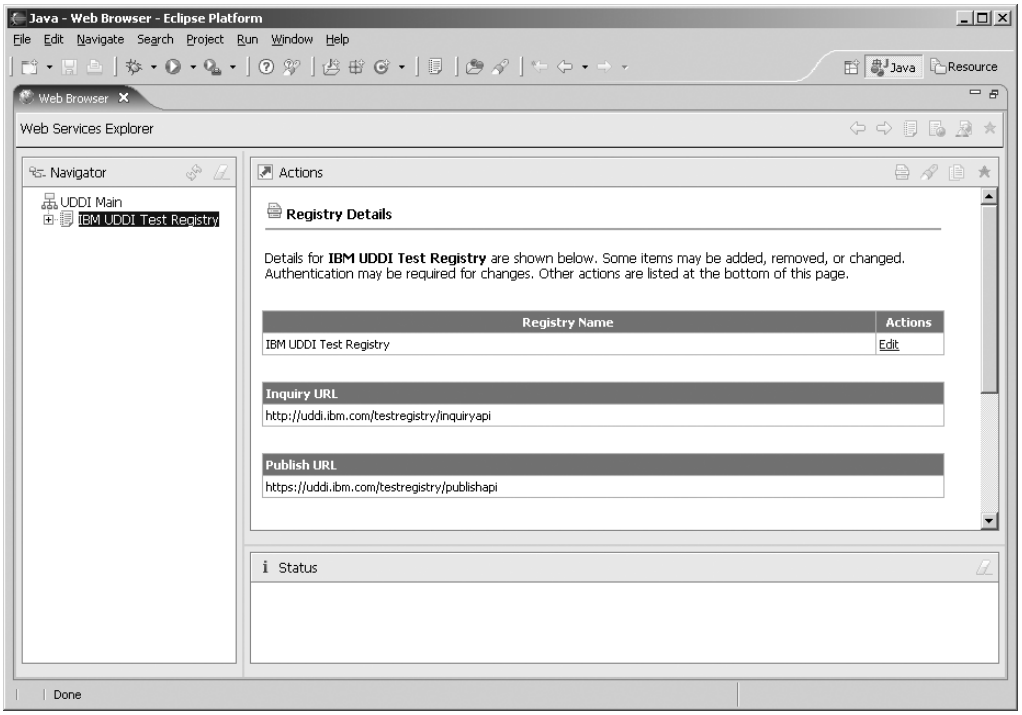

Figure 4-10. *Web Services Explorer*

Navigation Tools

The Navigation Tools comprise a series of views that are useful during the development phases of a J2EE application. These include the ability to navigate through the projects within the current workspace, view hierarchical contents of open resources, monitor problems with the module artifacts, track uncompleted tasks, monitor properties of module artifacts, view and manage instances for all servers defined for the project, observe the console messages generated by these servers, and also view the results of queries executed against data sources.

The J2EE Perspective and the Module View provides access to some of these navigation functionalities within Eclipse.

J2EE Perspective

A collection of several views useful for J2EE development activity make up the J2EE Perspective. Eclipse provides the concept of a perspective as a visual set of views and editors relating to a project type. This supplies the user with the ability to easily open up a series of related views that have relevance to a project being worked on. A J2EE web application, for instance, will often include multiple files that need to be modified, a server to deploy packaged artifacts to, and a console to view the output of the server. The J2EE Perspective provides instant accessibility to these useful views; Table 4-3 lists the available views.

Table 4-3. *Views from the J2EE Perspective*

View	Description
Project Explorer	A navigator provided to browse through the contents of open project. The Project Explorer is aware of J2EE components and denotes the different component types with appropriate icons.
Outline View	Displays the structural elements contained within a document being edited using a tree view structure, providing users with the ability to see what elements within the document relate to each other.
Problems View	A list of the problems existing with the current project being modified such as compilation and validation errors. The view contains a reference to the document and the specific line in which the error exists.
Tasks View	Provides a listing of items that have been delegated as *tasks*, which are often reminders to the developer. Task tags are defined in the Preferences window and support the ability to specify a priority level. Inclusion of this tag inside the comment of a document generates a reference within the view.
Properties View	Provides the ability to inspect and modify the properties of the current element selected within a document. The properties available differ based on the type of document being edited and the element type selected within such a document.
Servers View	Provides a mechanism to control the operations of all configured runtime instances of application servers.

It should be noted that in addition to the views that make up a perspective, additional views can be opened by selecting Window ➤ Show View ➤ Other and then choosing the needed view. The new view then opens alongside existing views from the J2EE Perspective, resulting in a customized perspective. Any such customized perspective can be saved by selecting Window ➤ Save Perspective As for later access.

Synchronization automatically occurs between each view and the source document in the editor. The Outline View, for instance, will immediately show the hierarchical outline of a document being edited, while the Properties View would show the property settings of the actual tag selected within the document.

Module View

This tool is available within the Project Explorer, providing a navigable representation of all the deployable J2EE artifacts that are available within a project. The contents available within the Module View match the elements specified within the deployment descriptor of a deployable module, and the deployment descriptor is in turn specific to a type of project. This means that a module for a Web Project will contain references for servlets, servlet mappings, etc., while a module for an EJB Project will contain references for session, entity, and message-driven beans. Figure 4-11 shows the Module View for a J2EE Web Project.

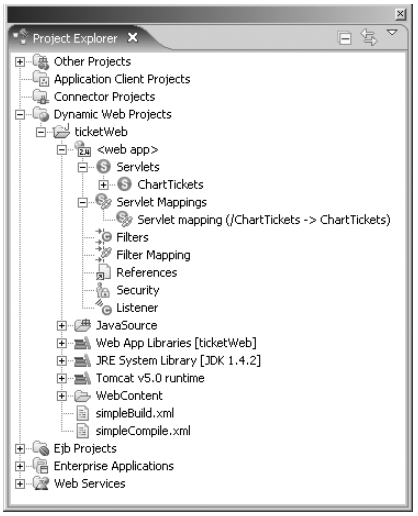

Figure 4-11. *Module View for a Web Project within the Project Explorer*

Summary

We began this chapter by providing an introduction to JST and describing its usefulness to both the enterprise application developer and tool extender. Next, we examined the scope of JST and discussed the criteria that determined which standards and technologies would be supported. The J2EE Core Model, or JCM, was next described, including details on its abstraction of core J2EE functionality and the APIs it exposes. This foundation, as we stated, is of great value to anyone, such as third-party vendors or open source developers interested in extending Eclipse to provide support for currently unsupported technologies.

Finally, we provided an introduction to the different useful tools that are included in JST and briefly discussed some of their uses and benefits. This discussion is expanded in later chapters of this text, providing detailed examples of using these tools in the context of implementing a project.